HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Lukas Neumann

Entwurf und prototypische Implementierung einer Analyse-Software für Massen-Messdaten für die Halbleiterfertigung

Dresden, 2010

Fakultät Mathematik/Naturwissenschaften/Informatik

DIPLOMARBEIT

Entwurf und prototypische Implementierung einer Analyse-Software für Massen-Messdaten für die Halbleiterfertigung

Autor:
Herr
Lukas Neumann

Studiengang:
Angewandte Informatik

Seminargruppe:
IF05wP1

Erstprüfer:
Prof. Dr.-Ing. Wilfried Schubert

Zweitprüfer:
Dipl.-Inf. Matthias Frank

Einreichung:
30.11.2010

Verteidigt am:
17.12.2010

Bibliografische Angaben:

Neumann, Lukas:
Entwurf und Prototypische Implementierung einer Analyse-Software für Massen-Messdaten in der Halbleitertechnik- 2010 - V, 62 S.
Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,
Fakultät Mathematik/Naturwissenschaften/Informatik, Diplomarbeit, 2010

Referat:

Die vorliegende Arbeit befasst sich mit der Entwicklung einer Analyse-Software für Massenmessdaten-Auswertung in der Halbleitertechnik. Das Hauptziel, eine leistungsstarke Lösung, auf Basis einer OLAP-Engine zu ermöglichen.

Inhaltsverzeichnis

Abbildungsverzeichnis IV

Tabellenverzeichnis V

1 Einleitung 1

1.1 Ziel der Arbeit 2

2 Theoretische Grundlagen 3

2.1 OLAP 3

2.1.1 Multidimensionaler-Würfel 3

2.1.2 Aggregation 5

2.1.3 Analyse der Evaluierungsregeln 6

2.1.4 Architekturkonzepte von OLAP-Systemen 9

2.1.5 OLAP-Funktionen 9

2.2 Halbleiterfertigung 10

2.2.1 Begriff Erklärungen 10

2.2.2 Prozessschritte 12

2.2.3 Qualitätssicherung durch Messungen 13

3 Analyse des Gegebenen 16

3.1 Messdaten-Spezifikation 16

3.2 Analyse der Aggregation Engine 16

3.2.1 SelectionModelList 17

3.2.2 Aggregation 18

3.2.3 ClusterData 18

3.2.4 Histogramm-Elemente 18

3.2.5 Filter 19

3.2.6 Transformation 19

3.2.7 Iterator 19

4 Anforderungs- und Problemanalyse 20

4.1 Entwicklungsplattform 20
4.2 Report Analyse 20
4.2.1 Report 20
4.2.2 Histogram-Report 23
4.2.3 Probabilty-Report 25
4.2.4 Statistic-Report 26
4.2.5 Simple-Report 27
4.2.6 CD-Uniformity-Report 28
4.3 Client Server Architektur 29
4.3.1 Grundlagen 31
4.3.2 Kommunkationsplattformen unter .NET 33
4.4 Antwortzeiten 33
4.4.1 Serialisierung 34
4.4.2 Zugriff auf aggregierte Daten 37
4.5 Automatisierte Aggregation 40
4.6 Einsatz von Datenbanken 41
4.6.1 ADO.NET 41
4.6.2 XML Datenbank 42
4.7 Mehrbenutzerbetrieb 42
4.8 Usability 42
4.8.1 Project-Reports-Konzept 43
4.8.2 Benutzeroberfläche 43
5 Entwurf 44
5.1 Software-Architektur 44
5.2 Datenhaltungsschichten 45
5.2.1 Aggregation Store 45
5.2.2 Cache Store 46
5.2.3 Project Store 47

5.3 Applikationsschicht 49
5.3.1 Cache Service 49
5.3.2 Aggregation Engine 54
5.3.3 Project Store Service 54
5.4 Präsentationsschicht 55
6 Zusammenfassung 57
Anhang 58
Literaturverzeichnis 62

Abbildungsverzeichnis

Abbildung 1 – Datenwürfel 4
Abbildung 2 – SEM Messung zur Kontrolle der Abbildungsqualität, Messung(links)-Design(rechts)... 14
Abbildung 3 – SelectionModelList 17
Abbildung 4 - Report 21
Abbildung 5 – DimensionModel 22
Abbildung 6 – Histogram Plot 24
Abbildung 7 – Histogram Plot 25
Abbildung 8 – Probability Plot 26
Abbildung 9 – Statistic-Table 27
Abbildung 10 - Simple Plot 28
Abbildung 11 – CDU Model 29
Abbildung 12 – Struct[] Test 36
Abbildung 13 – Class[] Test 36
Abbildung 14 - Dateigröße 37
Abbildung 15 – SimpleCache 39
Abbildung 16 – Project-Reports 43
Abbildung 17 - Architektur 45
Abbildung 18 – Aggregation Store 46
Abbildung 19 –CacheStore 47
Abbildung 20 – Project Store 48
Abbildung 21 –Project 49
Abbildung 22 – Cache Service 50
Abbildung 23 – Cache Status 51
Abbildung 24 – CallProcess 53
Abbildung 25 – Aggregation Engine 54
Abbildung 26 – Report Service 55
Abbildung 27 - AppFrame Übersicht 56
Abbildung 28 – Report Definition 56

Tabellenverzeichnis

Tabelle 1 – Ausgangstabelle-Tabelle 5
Tabelle 2 – Aggregations-Tabelle 6
Tabelle 3 – Arbeitsspeicherverbrauch 30
Tabelle 4 – CPU-Zeit 30
Tabelle 5 - Aggregationzeit 33
Tabelle 6 - Cubegröße 34

1 Einleitung

Thema dieser Arbeit ist die prototypische Implementierung einer Software, die Messdaten der Halbleiterindustrie analysiert. Die speziellen Herausforderungen der Datenbasis erfordern die Anwendung von hoch-performanten und flexiblen Analysemethoden. In dieser Arbeit, wird ausgehend von den Anforderungen eines speziellen Kunden, vor allem auf OLAP (Online Analytical Processing) basierte Methoden eingegangen.

Die Rahmenbedingungen für die Anwendung erfordern,

- die Antwortzeiten,
- Entwicklungszeiten,
- die Systemkosten und
- die Datenmenge

bei dem Systemdesign zu beachten.

Im Detail bedeutet dies:

- Die Lösung soll möglichst schnell die Messdaten für die Analysen bereitstellen, sodass die Messungen überprüft werden und die Analyse zur Prozessoptimierung herangezogen werden kann.
- Eine Lösung gesucht wird, die den Anforderungen entsprechen und deren Entwicklungszeiten Grenzen gesetzt werden.
- Die Wartungs- und Hardwarekosten begrenzt sind. Das System soll auf üblichen Desktop-Rechnern arbeiten.
- Es werden in wenigen Stunden Datenmengen bis zu 200 Millionen Datenzeilen, in Zukunft bis 5 Milliarden Datenzeile erzeugt, die statistisch aufbereitet werden müssen.
- Um die hier gesetzten Zeilen zu erreichen, muss eine detaillierte Analyse der Probleme und Anforderungen erfolgen.

Diese Arbeit und die Implementierung beachten, ohne immer darauf einzugehen, diese Rahmenbedingungen.

Eine weitere Herausforderung dieser Arbeit, auch für den Leser, ist die Begriffswelt und die Grundlagen der Halbleitertechnik. In einem Kapitel werden einige Verfahren des Halbleiterprozesses erklärt.

Die Implementierung erfolgte parallel zum Schreiben dieser Arbeit, war in Teilen bereits vor der Diplomarbeit vorhanden und wurde für die spezielle Anwendung überarbeitet. Prototypisch wurden vor allem solche Funktionalitäten implementiert, die wegen den Anforderungen besondere Herangehensweisen erforderten. Oft mussten Kompromisse gefunden werden, um technisch Machbares, technologisch Neues und Erprobtes auf die Anforderungen abzustimmen.

1.1 Ziel der Arbeit

Das Ziel dieser Arbeit besteht darin, für die bestehende OLAP-Engine neue Anforderungen sowie die Softwarestruktur zu erarbeiten und durch eine flexible serverbasierte Softwarestruktur umzusetzen.

Der Prototyp soll ein Grundgerüst bilden, dass später in die vorhandene DFMSim Architektur integriert werden soll.

Es soll eine, auf den Halbleitermarkt spezialisierte OLAP-Lösung erstellt werden, die wahrscheinlich in vielen Punkten von Open-Source-Anwendungen und anderen kommerziellen Anwendungen abweichen wird. Vor allem ist der Fokus nicht, im Gegensatz zu anderen Lösungen auf Seite der Datenanbindung gesetzt, sondern einem möglichst schnellem Antwortverhalten.

2 Theoretische Grundlagen

2.1 OLAP

Der Begriff OLAP (Online Analytical Processing) wurde 1993 durch den Datenbanktheoretiker Edgar Frank Codd geprägt. OLAP umfasst Technologien zur Ad-hoc-Analyse von großen Mengen an Daten, mit dem Ziel den Abruf von Informationen zu beschleunigen. Dabei ist OLAP nicht für die Verarbeitung von Transaktionen konzipiert, sondern zur Bereitstellung von Daten mit dem Ziel, Entscheidungsprozesse und Erkenntnisförderung in einem Unternehmen zu fördern [Kur99, 313-314]. Dazu werden die verarbeiteten Daten in einem hierarchischen oder nicht-hierarchischem mehrdimensionalen Datenwürfel abgelegt.

Die Datenbereitstellung erfolgt über externe Datenquellen. Um eine Daten-Integrität zwischen dem Datenwürfel und den Eingangs-Daten zu gewährleisten, müssen die Daten redundant sein. Hierzu werden die Daten in einem separaten System persistent abgelegt. An dieser Stelle kann es zu einer Datenaufbereitung kommen, um einen effizienten Zugriff zu ermöglichen. Oft werden für diesen Einsatz Datawarehouse Lösungen eingesetzt [Nag06, 225-226]. Hier übernimmt OLAP den Stellenwert der Applikationsschicht und bereitet die Daten, entsprechend der Anforderung des Benutzers, für die weitere Verwendung vor [Kur99, 188]. Wobei auch ein Data Mining Tool in Einsatz kommen kann. Im Gegensatz zu OLAP-Werkzeugen, welche der reinen Datenanalyse dienen, kann Data Mining unter anderem eine Prognose über Zusammenhänge, Muster und Trends der vorliegenden Daten abgeben. Data Mining kann auch unterstützend für OLAP Systeme eingesetzt werden, sodass für den Benutzer nur die signifikanten Stellen angezeigt werden [Bod06, 46].

2.1.1 Multidimensionaler-Würfel

Die zu analysierenden Daten werden in Fakten und Dimensionen unterteilt. Dabei werden mit Hilfe von Dimensionen Vektorräume gespannt, in welchen die Fakten angeordnet werden. Entsprechend der Anzahl der Dimensionen, kann bei drei Dimensionen ein Würfel entstehen. Werden mehr als drei gebildet, so spricht man von einem Multidimensionen-Würfel bzw. Hyper-Cube [Bod06, 40]. Die Kombination aus Dimensionen und Fakten werden als Multidimensionales Modell bezeichnet.

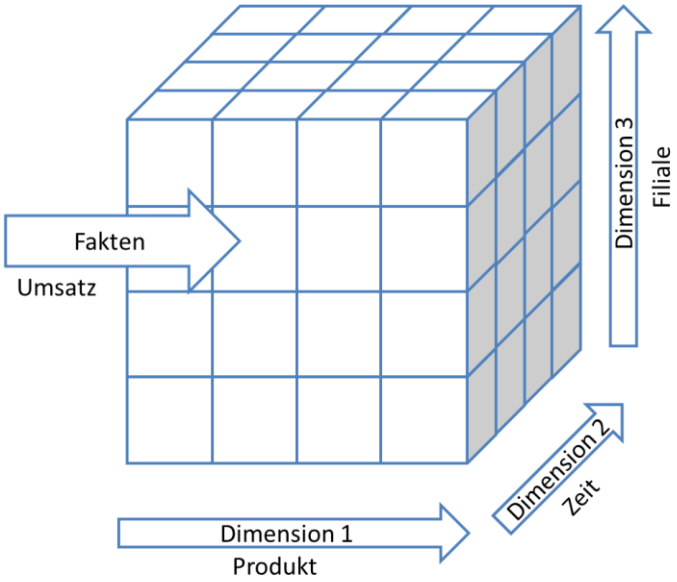

Abbildung 1 – Datenwürfel

Fakten

Bei Fakten bzw. Kennzahlen (engl. Facts/Measures) handelt es sich um numerische oder enumerische Datentypen, welche mindestens einer Dimension zugeordnet werden müssen. Unter enumerischen Datentypen versteht man eine Menge an Konstanten. Da Fakten nicht nur an eine Dimension gebunden sein müssen, können diese mit Berechnungsvorschriften aus anderen Fakten abgeleitet werden. Beispielsweise kann aus 2 Fakten, welche Verkaufszahlen repräsentieren, der Minimalwert abgeleitet werden [Kur99, 131-134].

Dimensionen

Dimensionen beschreiben die Sicht auf die Faktenwerte. Sie stehen somit im direkten Kontext zu den Kennzahlen und können dabei als hierarchisch betrachtet werden. Dies ermöglicht die Sicht der Fakten mit unterschiedlichen Aggregationsdichten [Kur99, 139].

Notation

Um eine einheitliche Form für den Verlauf der Arbeit zu gewährleisten, wird hierzu die folgende Notation für die Beschreibung eines Dimension Modells eingeführt:

> Dimensionen werden mit ⊗ voneinander getrennt in Klammern gesetzt und Fakten werden durch den Präfix ~ gekennzeichnet.

Für das in Abbildung 1 dargestellt Beispiel würde das Dimensions Modell wie folgt aussehen:

$$(\text{Produkt} \otimes \text{Zeit} \otimes \text{Filiale}) \sim \text{Umsatz}$$

2.1.2 Aggregation

Mit Hilfe der definierten Dimensionen werden Tupel gebildet. Diese liegen über den zu aggregierenden Datensatz, welche auf eine oder mehrere Tabellen verteilt liegen können. Ein Tupel stellt dabei einen Vektor zu einem Fakt dar. An der Position können sich keine oder mehrere Datensätze befinden [Kur99, 357]. Diese Datensätze können über eine oder mehrere Funktionen aggregiert werden. Das bedeutet, dass die Daten unter bestimmten mathematischen Vorschriften zusammengefasst werden und zu einem gewollten Datenverlust führen. Dabei können die Statistischen-Berechnungen, Histogramme sowie Statistische-Tests zur Aggregation beitragen [Tho02, 556-558].

Aggregation mittels einer Tabelle

Dieser recht komplexe Vorgang soll an dieser Stelle mit einer vereinfachten Form der Aggregation dargestellt werden. Hierzu kommen fiktive Ausgangsdaten und auf den in 2.1.1 vorgestellten Modell zur Anwendung.

Tabelle 1 – Ausgangstabelle-Tabelle

Index	Stadt	Filiale	Produkt	Zeit (Monat)	Umsatz
1	München	Filiale M	Weißbier	1	100
2	Bamberg	Filiale M	Weißbier	1	300
3	Dresden	Filiale B	Pils	1	200
4	Dresden	Filiale B	Pils	1	100
5	Weixdorf	Filiale B	Weißbier	1	500
6	Dresden	Filiale R	Pils	2	200

Die Aggregation findet auf dem vorliegenden Modell statt:

$$(\text{Produkt} \otimes \text{Zeit} \otimes \text{Filiale}) \sim \sum Umsatz$$

Betrachtet man Tabelle 1 so kann man die Stadt und den Index ignorieren, da sie in keiner Dimension oder als Fakt vorkommen. Filiale, Produkt und Zeit sind die zu betrachtenden Dimensionen und die Aggregation erfolgt auf der Summenfunktion über den Umsatz.

Anstelle Tupel über die Dimension zu bilden, werden die Zeilen einzeln betrachtet und in einer separaten Tabelle niedergeschrieben. Beim Einfügen jeder neuen Zeile werden diese, mit den Zeilen aus dem Vorjahr, auf Gleichheit der Dimensionen überprüft und es muss auf dem Umsatz die Summenfunktion angewendet werden. Dies würde auch einem Vektor entsprechen, der auf

mehrere Fakten zeigt. In Zeile 1 und 2 sowie 3 und 4 sind die Dimensionen gleich. In Tabelle 2 wird das Ergebnis der Aggregation dargestellt.

Tabelle 2 – Aggregations-Tabelle

Filiale	Produkt	Zeit (Monat)	$\sum$ *Umsatz*
Filiale M	Weißbier	1	400
Filiale B	Pils	1	300
Filiale B	Weißbier	1	500
Filiale M	Plis	2	200

2.1.3 Analyse der Evaluierungsregeln

Mittels von Evaluierungsregeln werden Anforderungen an ein OLAP-System definiert.

2.1.3.1 Die 12 OLAP-Grundregeln von Edgar F. Codd

Edgar F. Codd hat 1993 im Magazin Computerworld seine zwölf Regeln zur Evaluierung einer OLAP-Anwendung vorgestellt. Diese Regeln sind im Rahmen seiner Tätigkeit bei der Firma Arbor entstanden. Es war natürlich abzusehen, dass nur die Arbor's Produkte die Regeln zu diesem Zeitpunkt erfüllt haben. Es führte dazu, dass die Publikation sehr in Misskredit geraten ist [Tho02, 615]. Trotzdem stellen diese Regeln interessante Anforderungen:

1. Multidimensionale konzeptionelle Sicht

   Die multidimensionale konzeptionelle Sicht auf die Daten stellt das wichtigste Kriterium an eine OLAP-Anwendung. Dies soll dem Benutzer die Möglichkeit geben, vorliegenden Daten aus unterschiedlichen Sichtweisen zu analysieren.

2. Transparenz

   Transparenz ist die Trennung von der Benutzerschnittstelle und der zu Grunde liegenden Architekturen des OLAP-Systems. Dabei soll zum einen Komplexität der im System vorliegenden Vorgänge dem Benutzer verborgen werden. Zum anderen kann mit einer Trennung eine Integration in andere Werkzeuge geschaffen werden.

3. Zugriffmöglichkeiten

   Da es sich bei der OLAP Engine um eine Middleware-Lösung handelt, welche sich zwischen der Datenhaltungsschicht und einem Frontend befindet [Nag06, 206], muss die Engine in der Lage sein mit unterschiedlichen Quellen zu operieren. Um dies zu gewährleisten muss eine leicht integrierbare Konvertierungsmöglichkeit geschaffen werden.

4. Konsistente Leistungsfähigkeit

   Bei der Berichterstattung ist zu gewährleisten, dass unabhängig von der Datenmenge und der Komplexität der Analyse Leistungseinbußen für den Benutzer nicht ersichtlich sind.

5. Client/Server Architekturen

   Mit einer Client-Server-Architektur soll eine möglichst optimale Lastenverteilung und Performance der Berichterstattung realisiert werden.

6. Generische Dimensionen

   Alle Dimensionen sollen in Bezug auf Ihre operative Funktionalität und logische Struktur in einer einheitlichen Form vorliegen.

7. Dynamische Handhabung von dünnbesetzten Matrizen

   Bei dünnbesetzten Matrizen (engl. Sparse-Matrix) handelt es sich um Matrizen, welche eine so hohe Anzahl an Leerstellen oder gleichen Werten aufweisen, dass es sich lohnt, diese gesondert zu behandeln [Pis84, 1-2]. Sie entstehen verstärkt bei Datenwürfeln mit einer hohen Dimensionalität, da die Anzahl der Datensätze konstant ist aber die Anzahl der Zellen zunimmt. Dieses Verhalten führt zu einer Leistungsminderung des Systems und muss durch das OLAP-System behandelt werden.

8. Mehrbenutzerunterstützung

   Das OLAP-System muss in der Lage sein, Parallelzugriffe mehrerer Benutzer verwalten zu können. Dazu müssen konkurrierende Zugriffe, Integrität und Sicherheit der Daten, mit einem Transaktionsmechanismus gesteuert und gewährleistet werden. Diese Regel steht auch in Übereinkunft mit der Client/Server Architektur.

9. Unbeschränkter dimensionsübergreifende Operationen

   Das System soll auf allen Dimensionen Berechnungsoperationen uneingeschränkt unterstützen.

10. Intuitive Datenanalyse

    Eine Datenanalyse erfordert den Eingriff des Benutzers. Dazu müssen einfache Navigations- und Orientierungsmöglichkeiten durch das Front-End geboten werden.

11. Flexible Berichterstattung

    Die Berichterstattung soll hinsichtlich der Anforderungen an die Analysen flexibel sein. Dabei sollen Berichte, in Abhängigkeit der Datenbasis, frei definierbar sein.

12. Unbegrenzte Anzahl an Dimensions- und Aggregationsebenen

    Hinsichtlich der Anzahl an Dimensionen sowie Aggregationsebenen, innerhalb einer Dimension, dürfen keine Einschränkungen durch das OLAP-System vorliegen. Da dies, insbesondere für die Anzahl der Dimension, aus technischer Hinsicht nicht möglich ist, wurde eine Dimensionsanzahl auf 15 bis 20 reduziert.

Diese Regeln sind sehr Produktspezifisch und unterscheiden nicht zwischen technischen Ansprüchen und Anforderungen an eine Softwarelösung. Einige Regeln stellen aber interessante Erwartungen an ein Produkt zur Messdaten-Analyse dar.

2.1.3.2 Die FASMI-Definition von Pendse und Creeth

Die zwölf Regeln von Codd waren bei seinen Kollegen sehr umstritten, da sie zu produktspezifisch waren. Deshalb haben Pendse und Creeth eine allgemeine Definition für das OLAP-Konzept erarbeitet, welches als der FASMI-Test bekannt ist [WWW02].

1. Fast

   Die Anfragen an das System sollen möglichst schnell abgewickelt werden. Einfache Abfragen nach Möglichkeit nicht länger wie eine Sekunde andauern. Eine komplexere Abfrage darf maximal 20 Sekunden in Anspruch nehmen. Zu lange Wartezeiten würden beim Benutzer den Verdacht erwecken, dass die Anwendung nicht mehr reagiert, unabhängig von einer Information an den Benutzer.

2. Analysis

   Hierzu ist es notwendig, dass umfangreiche Analysemethoden aus betriebswirtschaftlichen und statistischen Bereich dem Benutzer zur Verfügung gestellt werden. Dabei sollen Analyseabfragen ohne Vorkenntnisse einer Programmiersprache durchzuführen sein.

3. Shared

   Mehrbenutzerbetrieb muss durch OLAP gewährleistet werden. Hierzu sind unter anderem Sicherheitsmechanismen für lese und schreib Operationen zur Verfügung zu stellen.

4. Multidimensional

   Wie bei Codd stellt die Multidimensionalität ein wichtiges Kriterium an OLAP. Auswertung von mehrdimensionalen Modellen, sowie vollständige Unterstützung von Hierarchien sind als essentiell zu bewerten. Pensen und Creeth haben an dieser Stelle keine minimale Anzahl an Dimensionen vorgeschrieben.

5. Information

   Das System muss in der Lage sein, große Datenmengen zu verwalten. Hierzu muss der Zugriff auf alle Daten und die dazugehörigen Informationen, unabhängig der Datenquelle, dem Benutzer zu Verfügung gestellt werden.

2.1.4 Architekturkonzepte von OLAP-Systemen

Der Begriff OLAP-Systeme wird meist auf reine Datenbanktechnologien reduziert. Unterteilt können OLAP-Systeme in relationale und multidimensionale Ansätze, die als Konzepte der Datenhaltung bezeichnet werden.

Die beiden Ansätze werden in folgende Konzepte unterschieden [Kur99, 327-334]:

- ROLAP: Relational OLAP

  Der Datenzugriff erfolgt auf einer relationalen Datenbank, dabei wird mittels einer SQL-Erweiterung die Aggregation durchgeführt.
- MOLAP: Multidimensional OLAP

  Hierbei wird eine Multidimensional Datenbank verwendet, dabei handelt es sich um eine effiziente Strukturierung und Speicherung von großen Datenarrays.
- HOLAP: Hybrid OLAP

  Im HOLAP werden die Vorteile aus ROLAP und MOLAP zu einen System vereint.
- DOLAP: Desktop OLAP

  Es werden Teile des Aggregierens auf einem Desktop-System gespeichert.

2.1.5 OLAP-Funktionen

Bei OLAP-Funktionen handelt es sich um Methoden, welche eine Navigation innerhalb des Würfels erlauben. Zu den wichtigsten zählen die Folgenden [Kur99, 334-338]:

- Dripp-Down / Roll-Up

  Der Detailgrad der Daten auf den Dimensionen wird erhöht oder verringert. Dies ist mit Zoom in/out zu vergleichen.

- Pivot

  Die Dimensionen werden untereinander innerhalb eines Würfels vertauscht.
- Slice

  Ausschneiden einer Dimension aus dem Würfel
- Dice

  Ausschneiden von bestimmten Bereichen aus einem Datenwürfel

2.2 Halbleiterfertigung

Der Inhalt dieses Kapitels beruht auf Lehrunterlagen der Halbleitertechnik [Baum05] sowie aus firmeninternen Gesprächen.

2.2.1 Begriff Erklärungen

Los

Ein Los ist eine Produkteinheit,

- die als Einheit erzeugt (eingeschleust) wird
- von Prozessschritt zu Prozessschritt bewegt wird,
- als Einheit bearbeitet wird

Ein Los kann aus mehreren Teilen bestehen, in der Halbleiterfertigung üblicherweise aus 25 Wafer, die im Tool sequentiell/ teilweise in Gruppen parallel abgearbeitet werden.

Chip/ Halbleiterbaustein

Ein Chip ist ein Halbleiterelement, das bestimmte elektronische Eigenschaften und Funktionalitäten besitzt. Chips findet man heute überall:

- CPU
- Speicherbausteine
- Sensoren
- Logikbausteine (z.B. in Uhren, Mobiltelefonen)
- Passive Elemente, wie Chips auf Kreditkarten und typische Sicherheitstags auf Produkten

Die

Ein Die ist ein Halbleiterchip während der Produktion. Ein Die enthält neben den Chip spezifischen Funktionalitäten weitere Teststrukturen und elektrisch aktive Elemente, die während der Produktion zum Messen und Testen notwendig sind. Daher gibt es eine Unterscheidung zwischen Chip und Die.

Wafer

Ein Wafer ist eine runde Silizium- oder Glasscheibe, die als Träger der Dies dient. Auf dem Wafer werden durch verschiedene Prozessschritte Material aufgetragen und strukturiert. Nach der Produktion, im Backend, wird der Wafer von der Unterseite geschliffen, sodass nur noch die Dies auf der Oberseite bestehen bleiben. Dieser sehr dünne Wafer wird dann in die einzelnen Dies zersägt bzw. geritzt und gebrochen.

Field

Das Field ist ein sich wiederholender Bereich auf dem Wafer, der mehrere Dies beinhalten kann und durch einen Lithografischen Belichtungsschritt erzeugt wird.

Kerf

Die Kerf ist ein Bereich zwischen den Dies. Dieser Bereich kann Teststrukturen, Mess-Targets und andere Information enthalten. Sie ist nur während der Produktion notwendig und wird später beim Trennen der Dies zerstört.

Target bzw. Mess-Pattern

Währen der Produktion wird das Los bzw. der Wafer sehr oft gemessen und getestet. Beim Messen werden an Target bestimmte Masse gewonnen (z.B. Verschiebung zur Vorebene). Target- und Mess-Pattern unterscheiden sich insofern, dass Mess-Targets vordefinierte, nur für die Messung vorhandene Strukturen bilden, die meist im Kerf-Bereich liegen. Mess-Pattern allgemein kann sich auf beliebige Strukturen beziehen, auch auf funktionale Strukturen innerhalb des Dies, die gemessen werden.

2.2.2 Prozessschritte

Der Halbleiterprozess besteht aus mehreren hundert Einzelschritten. Jeder Schritt ist sehr komplex. Es kann daher nur kurz eine grobe Einführung gegeben werden.

1. Oxidation

   Eine dünne Schicht von Siliziumdioxid wird auf den Wafer aufgetragen. Diese Schicht wird in weiteren Prozessschritten strukturiert.

2. Belackung

   Beim Lackvorgang wird der Wafer mit einem fotoempfindlichen Lack benetzt. Hierzu wird der Wafer waagerecht um seinen Mittelpunkt zum Rotieren gebracht und eine dosierte Menge Lack punktiert aufgebracht. Die bei der Rotation entstehenden Fliehkräfte verteilen die Lösung gleichmäßig auf der Scheibe. Dies gewährleistet eine gleichmäßige Schichtdicke, welche durch die Drehzahl und Konsistenz des Lackes gesteuert werden kann. Bei diesem Verfahren spricht man von Schleuderlackierung oder auch Spin-Coat-Verfahren.

3. Belichtung

   Mit Hilfe einer Maske (eng. Retikel), welche ein vergrößertes Abbild einer Strukturschicht darstellt, wird der Wafer belichtet. Man spricht an dieser Stelle auch von Lithographie. Zwei Verfahren sind hier hervorzuheben: die Elektrostrahlen- und Photolithographie. Die Belichtung kann durch einen Stepper schrittweise (stepping) oder durch einen Scanner fließend (scan) erfolgen.

4. Entwicklung

   Abhängig vom Lack, man unterscheidet zwischen positiv und negativ, werden entweder die belichteten oder unbelichtete Stellen vom Lack befreit. Nach diesem Prozessschritt ist entweder das verkleinert Abbild oder negativ des Retikels auf den Wafer zu sehen.

5. Ätzen

   Beim Ätzen wird die lackfreie Materialschicht abgetragen, die vom Lack geschützte Schicht bleibt geschützt.

6. Dotieren

   Durch die Dotierung wird die elektronische Leitfähigkeit des Materials geändert, man unterscheidet zwischen p und n-Dotierung. Hierzu werden für die p-Dotierung Fremdatome

und die n-Dotierung Elektronen implantiert. Auch hier werden die lackgeschützten Stellen von der Dotierung nicht beeinflusst.

7. Metallisierung
   Eine Metallschicht (Leiterbahnen) wird auf den Wafer ausgetragen. Dieser Vorgang erfolgt durch Bedampfen mit dem Metall.

8. CMP
   Beim chemisch-mechanischen Planarisieren (engl. chemical mechanical planarization) wird der Wafer zum einen von den Lackresten gereinigt und zum anderen poliert. Beim Polieren werden durch regulierten Druck Unebenheiten auf dem Wafer angepasst. Eine Regulierung der Höhe kann mit diesem Verfahren ebenfalls erzielt werden.

9. Wiederholung
   Die Prozesse 1-8 werden entsprechend der Komplexität des Produktes und der Schichtenanzahl ca. 30-mal wiederholt.

10. Abschluss
    Schließlich werden die einzelnen Chips aus dem Wafer ausgesägt, Verbindungen angelegt und in ein Chip-Gehäuse (engl. Package) eingeschlossen.

2.2.3 Qualitätssicherung durch Messungen

Bei den oben genannten Produktionsschritten kann es aufgrund von Äußeren und Produktionsabhängigen Einflüssen zu Fehlern kommen. Abbildung 2 zeigt einen solchen Fehler, die Design-Daten (recht) entsprechen nicht den Messdaten (links), insbesondere in den mittleren Feld.

Diese müssen möglichst schnell erkannt werden, so dass eine Korrektur im laufenden Prozess durchgeführt werden kann. Hier werden exemplarisch zwei übliche Messungen vorgestellt, die die Qualität der Lithografie sicherstellen.

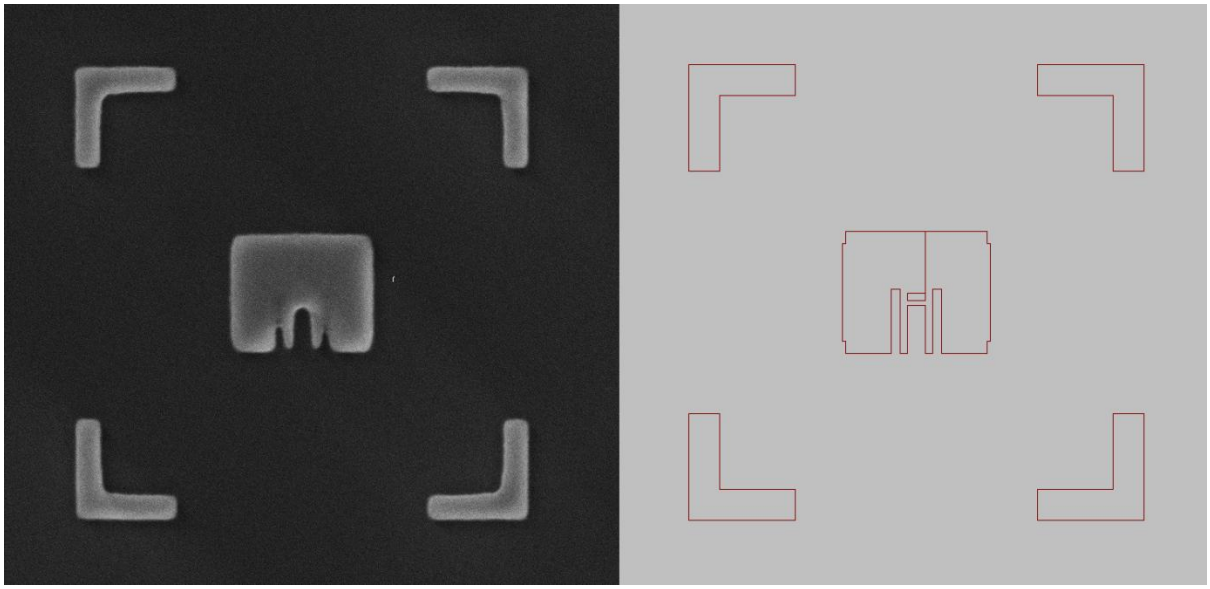

Abbildung 2 – SEM Messung zur Kontrolle der Abbildungsqualität, Messung(links)-Design(rechts)

2.2.3.1 Overlay Messung

Bei der Overlay Messung werden die Messstrukturen innerhalb der Kerf eines Feldes angefahren und mit den Strukturen aus den Vorebenen verglichen. Es werden aus zeitlichen Gründen, insbesondere in der Produktionslinie, nicht alle Felder bemessen. Dieses wird auch als Sampling bezeichnet, da nur eine repräsentative Menge gemessen wird. Sampling findet nicht nur auf der Ebene des Wafer statt, sondern auch bei einem Los. In Ausnahmefällen kann auch eine 100%-Messung erfolgen, in der alle Felder bemessen werden. In den meisten Fällen werden 20-40 Messungen pro Wafer vorgenommen, dabei erfolgt die Messung meist bei 3 Wafern pro Los. Die relativ kleine Anzahl an Messungen reicht jedoch aus, um eine Aussage über die Qualität der Prozessschritte abzugeben.

Dabei können drei mögliche Fehler erkennbar sein:

- Rotation
- Magnification
- Translation

Mit einer modellbasierenden Hochrechnung kann eine Analyse über den kompletten Wafer abgegeben werden. Diese wird auf das Parametermodell des Tools und der Justierung eines Produktionswerkzeugs abgebildet. Die dabei entstehenden Erkenntnisse werden dazu benutzt, um mit einer neuen Konfiguration (engl. Setup) der Tools, den Fehlern entgegen zu lenken. Hierbei spricht man von Run-To-Run (R2R), dabei wird von Messung zu Messung eine Korrektur vorgenommen.

2.2.3.2 CD Messung

Die Critical-Dimension-Messung misst üblicherweise Strukturen innerhalb des Dies, alternativ Teststrukturen in der Kerf die versuchen die Strukturen nachzubilden.

Die CD Messung soll sicherstellen, dass die Lithografie und der Ätzschritt die spezifizierten Masse der Strukturen erreichen. Ist dies nicht gegeben, ändern sich wesentliche elektronische Eigenschaften wie Schaltzeiten, Widerstand und Kapazitäten. Dies führt zu höheren Stromverbrauch, mehr Abwärme oder zum Ausfall des Chips.

3 Analyse des Gegebenen

3.1 Messdaten-Spezifikation

Die Messdaten werden mit einer CSV-Datei (Character Separated Values) für die Analyse bereitgestellt. Die CSV-Datei besteht dabei aus der Kontextinformationen und den Mess- und Designdaten.

Kontextinformationen

Die Kontextinformationen geben Auskunft über die durchgeführte Messung. Dies beinhaltet unter andere:

- Zeitpunkt der Messung
- Losnummer
- Wafer innerhalb des Los
- Angaben zum Wafer
- Angabe zu Messmethode- und Parametern

Messdaten

Messdaten sind Daten, welche aktiv gemessen worden sind z.B.: die Breite einer Struktur und jene, die mit einer Berechnungsvorschrift aus aktiven Messdaten und Designdaten entstanden sind.

Designdaten

Bei Designdaten handelt es sich zum einen um Daten, die von externen Quellen kommen, wobei hier unter anderem Informationen aus CAD[1]-System zur Anwendung kommen.

3.2 Analyse der Aggregation Engine

Die vorliegende Aggregation Engine wurde durch die Firma DFMSim GmbH speziell für die Anforderungen in der Halbleiterindustrie entwickelt. Hierzu zählt die Bearbeitung von einer extrem großen Anzahl an Messpunkten in möglichst kurzer Zeit.

Die zu bearbeitenden Messwerte werden in der vorliegenden Version sequenziell aus einer CSV Datei gelesen und auf Basis einer SelectionModelList verarbeitet. Das Resultat ist ein aggregiertes Abbild der Messwerte. Üblicherweise erfolgt die Verarbeitung in zwei Schritten, in dem die Daten gefiltert und an die Aggregations-Engine weitergeleitet werden. Dabei entstehen ein oder mehrere

[1] Computer-Aided-Design (rechnerunterstützer Entwurf)

Daten-Cubes, deren Zellen aus Histogramm-Elementen zusammengesetzt sind. Die so erstellten Cubes können mit Transformationen im Sinne von OLAP-Operationen manipuliert werden. Da sich die Handhabung der resultierenden Datenwürfel als sehr schwierig gestalten kann, wurde für diese Zwecke im Laufe der Arbeit eine Iterator-Hilfsklasse entwickelt.

3.2.1 SelectionModelList

Die SelectionModelList (Abbildung 3) ist als Konfigurationsträger für die Aggregation-Engine zu betrachten. Eine vollständige SelectionModelList besteht aus einem oder mehreren SelectionModel und beinhaltet die eigentliche Konfiguration für die auszuführende Aggregation. Diese besteht aus 3 Komponenten:

- Source

  Besteht aus einer SelectionSource, des Messdateneingang.
- Items

  Hierbei handelt es sich um ein Array aus CoreGroupTypes. Es folgt eine Aufteilung in zwei Gruppen FilterTypes und GroupingTypes. FilterTypes sind für die Filterung von Zeilen und GroupingTypes für die Gruppierung der Dimensionswerte zuständig.
- ClusterAnalysis

  In ClusterAnalysis wird eine Dimension angegeben, über welche eine Zusammenfassung der Daten erfolgt. Für den Bereich der Messdatenanalyse setzt man an dieser Stelle einen Messwert ein.

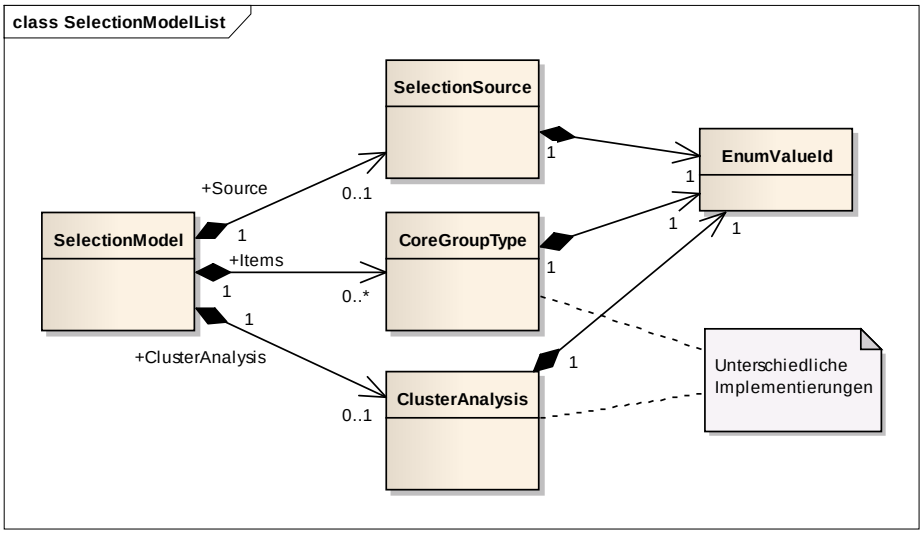

Abbildung 3 – SelectionModelList

Alle Komponenten haben gemeinsam, dass sie das Attribut EnumValueId besitzen, welches eine Zeile innerhalb der CSV Datei repräsentiert.

3.2.2 Aggregation

Bei der Aggregation entstehen, mit Hilfe der im Dimension-Modell definierten Dimension und Fakten, Datenwürfel mit den zusammengefassten Messwerten. Hier bestehen die Fakten jedoch nur aus einer Dimension und sie können nur nummerische Datentypen beinhalten. Nach der Aggregation ist es nicht gewährleistet, dass die erstellten Cubes mit dem Aggregations-Modell übereinstimmen. Dies kann bedeuten, dass aus einem einfachen Modell mehrere Hyper-Cubes entstehen können. Auch die Reihenfolge der Dimensionen innerhalb des Cubes muss nicht mit der Reihenfolge im Dimensions-Modell übereinstimmen. Eine hierarchische Aufteilung der Dimensionen wird an dieser Stelle nicht unterstützt. Dies kann jedoch durch die Angabe von Bereichen an den Dimensionen und Fakten erfolgen.

3.2.3 ClusterData

Das Resultat der Aggregation ist ein ClusterData-Array, die Anzahl der ClusterData hängt von der DimensionModelList ab. Ein ClusterData beinhaltet die bei der Aggregation verwendeten Dimensionen, diese werden als Scales bezeichnet und bestehen aus ScaleItems. Jedes Item repräsentiert den Bereich einer Dimension, in welcher es gruppiert worden ist. Mittels einer Kombination von Items aus unterschiedlichen Dimensionen kann auch das Histogramm-Element der Aggregation zugegriffen werden. Dieses beinhaltet die eigentliche Aggregation.

3.2.4 Histogramm-Elemente

Um die Datenmenge bei und nach der Aggregation möglichst gering zu halten, werden die Fakten in einem Histogramm-Elementen gesammelt. Histogramme werden oft zur Darstellung der Häufigkeit von Daten verwendet. Dazu werden auf der Abszisse (x-Achse) die Ausprägungen und auf der Ordinate (y-Achse) die Häufigkeit der Variablen in Säulen dargestellt [Ras06, 6]. Da in Histogrammen nur die Häufigkeiten innerhalb einer Breite und nicht die Daten direkt gespeichert werden, eignen sich diese zur Datenaggregation. Histogramme können bei übereinstimmenden Säulenbreiten zusammengeführt werden.

Das Histogramm führt über die gesammelten Werte zusätzlich eine Statistik. Diese beinhaltet die folgenden Werte:

- Minimum
- Maximum
- Anzahl

- Summe
- Quadratsumme
- Breite
- Standardabweichung
- Mittelwert

Breite, Standardabweichung sowie der Mittelwert werden nicht beim Sammeln, sondern mit Hilfe der anderen Werte ermittelt. Das hat den Vorteil, dass eine Zusammenführung von Statistiken möglich ist.

3.2.5 Filter

Filter dienen der Reduzierung der zu bearbeitenden und resultierenden Datenmenge. Sie werden vor dem Einfügen von Fakten, in das ClusterData, auf die zu bearbeitende Zeile der CSV Datei angewendet. Trifft ein Filter zu, wird diese Zeile aus der Aggregation ausgeschlossen.

3.2.6 Transformation

Mit Unterstützung der Transformationen kann ein Datenwürfel manipuliert werden. Hierzu steht dem Benutzer eine Reihe an Operationen zu Verfügung, die sowohl auf Dimensionen als auch auf Fakten verwendet werden können. Die Operationen entsprechen nicht denen innerhalb eines OLAP-Systems, stellen jedoch eine Teilmenge dar.

3.2.7 Iterator

Beim Iterator handelt es sich um eine Erweiterung des ClusterData-Arrays. Mit diesem wird ein Zugriff auf die im ClusterData beinhalteten Histogrammen erleichtert. Der Iterator liefert CdIteratorItems, diese bestehen aus Achsenbeschriftungen und Histogrammen. Achsenbeschriftungen entsprechen den Scales. In laufe der Entwicklung wurde der Iterator soweit erweitert, dass er mit den später eingeführten DimensionModel betrieben werden kann.

4 Anforderungs- und Problemanalyse

In diesem Kapitel werden die Anforderungen an eine Analysesoftware bestimmt, die aus allgemeinen Anforderungen, speziell für den Halbleiterbereich, bestehen. Die daraus entstandenen Bedingungen und Probleme werden diskutiert und gelöst.

Mittels der FASMI-Regeln nach Pendse und Creeth kann eine allgemeine Anforderungsliste an eine OLAP-basierende Analysesoftware gestellt werden. Es ist zu beachten, dass im vorliegenden bereits eine Aggregations-Komponente in Form der Aggregation Engine bereits vorhanden ist. Eine Berücksichtigung aller Evaluierungsregeln wird somit nicht vorgenommen, da diese bereits durch die Engine erfüllt werden.

4.1 Entwicklungsplattform

Die vorgegebene Aggregation-Engine wurde mit dem .NET Framework 3.5 entwickelt. Bei .NET-Framework handelt es sich um eine moderne Software-Plattform der Firma Microsoft. Es enthält eine große Anzahl an Bibliotheken, welche den Prozess der Implementierung vereinfachen und somit beschleunigen.

4.2 Report Analyse

Neben den allgemeinen Anforderungen an eine Analysesoftware, haben sich mit Unterstützung der firmeninternen Gespräche, sowie Kundengespräche, spezielle Anforderungen bezüglich der zu erstellenden Analysen ergeben. Diese sollen mit Hilfe von Berichten (engl. Reports) visuell dargestellt werden.

4.2.1 Report

Ein Report (Abbildung 4) wird durch ein DimensionModel sowie ein ReportTemplate definiert. Hierbei ist der Benutzer verpflichtet, ein ReportTemplate seinem Report zuzuweisen. Wird einem Report der Typ ProbabilityTemplate zugewiesen, so spricht man von einem Probability-Report.

DimensionsModel

Die derzeitige Version der Aggregation-Engine benötigt für die Ausführung eine SelectionModelList. Diese hat den Nachteil, dass sie als Konfigurationsmittel für einen Benutzer nicht geeignet ist, da nicht die Dimensionen beschrieben werden sondern der Bearbeitungsweg.

Eine allgemeine Anforderung an die Berichte ist eine freie Vergabe und Konfiguration von Dimension und Fakten an Achsen, Spalten oder Diagrammen[2]. Damit wird den Benutzern die Möglichkeit gegeben, Reports entsprechend den Prozess-Anforderungen zu definieren und analysieren. Zusätzlich dazu sollen die einzelnen Berichte, betreffend den darzustellenden Inhalt, konfigurierbar sein.

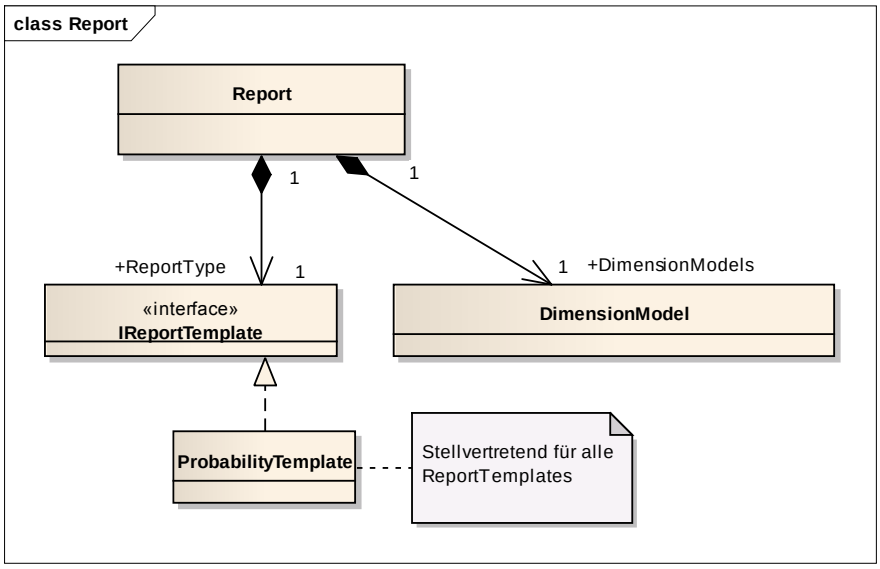

Abbildung 4 - Report

[2] In der Anwendung sowie EnumDimensionClass als Series bezeichnet

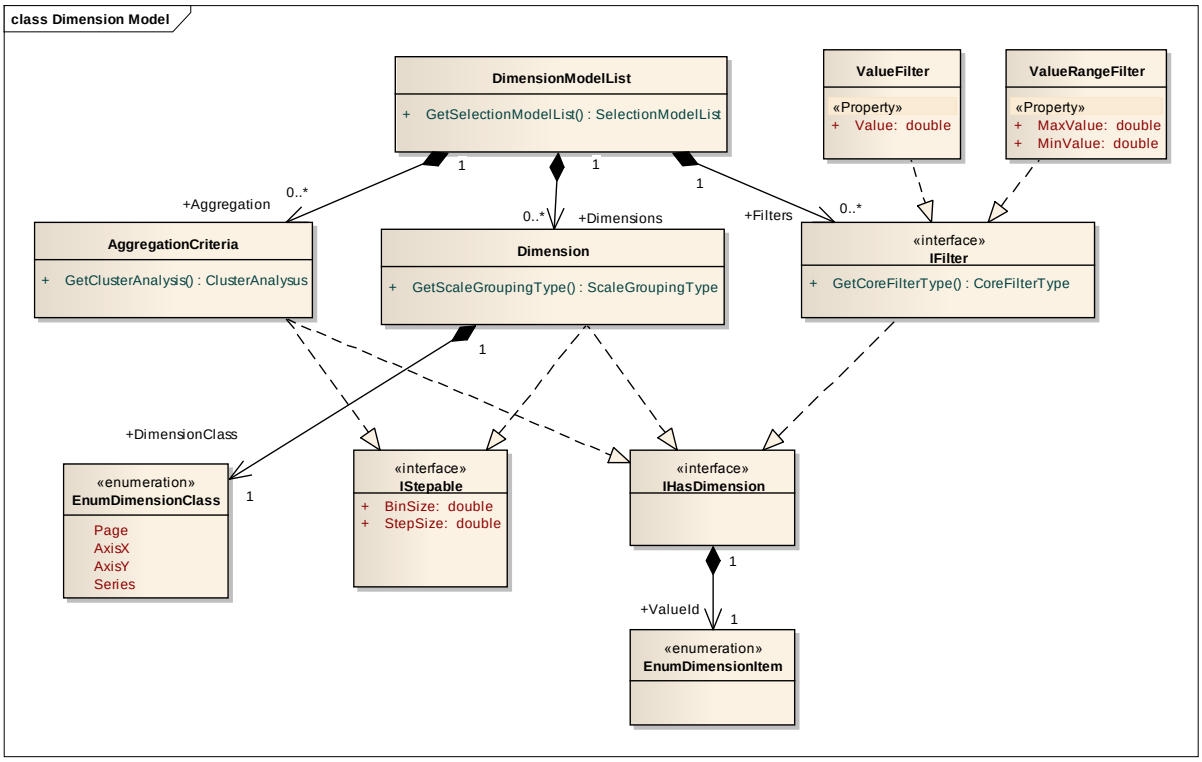

Abbildung 5 – DimensionModel

Vergleicht man die Abbildung 3 und Abbildung 5 so erkennt man an drei Stellen einen bedeutenden Unterschied.

- Kein Datenquelle

  Eine Datenquelle wird bei DimensionModel nicht benötigt, da diese nur das zu aggregierende Modell beschreibt und nicht den kompletten Bearbeitungsweg. Die Datenquelle wird erst durch den Server angebunden.
- Filter werden als Eigenschaft abgebildet

  Filter haben eine komplett andere Bedeutung in der Bearbeitung wie die Dimension-konfigurationen. Durch Filter werden nicht Dimensionen[3] verarbeitet, sondern Zeilen innerhalb des Kreuzprodukts der Dimensions-Vektoren.
- Zusätzliche Klasse für eine Dimension Klassifizierung

[3] Spalten

Dimensionen besitzen eine DimensionClass, diese dient einer eindeutigen Beschreibung, an welcher Stelle sich die Dimension innerhalb eines Reports befindet. Mit dieser Eigenschaft wird der SelectionModelList optimiert.

Ein weitere Unterschied ist, die Einführung des EnumDimensionItems, es soll nicht, wie das ValueIdx, direkt an einer Dimension gebunden sein, sondern kann stellvertretend mehrere Dimensionen zusammenfassen. Dadurch ist es möglich ein Model in einer einfachen Weise zu beschreiben.

$$(Width \otimes All_{Spaces}) \sim DW$$

Das oben beschriebene Modell steht hier stellvertretend für die folgenden SelectionModels:

$$(Width \otimes Space_1) \sim DW$$

$$(Width \otimes Space_2) \sim DW$$

$$(Width \otimes Space) \sim DW$$

$$(Width \otimes Space_{Symetric}) \sim DW$$

$$(Width \otimes Space_{Asymetric}) \sim DW$$

$$(Width \otimes Space_{Min}) \sim DW$$

$$(Width \otimes Space_{Max}) \sim DW$$

Die Einführung des DimensionModel bringt zwar keinen funktionalen Vorteil für die Anwendung, erleichtert und vereinheitlicht aber die Erstellung von Modellen.

4.2.2 Histogram-Report

Der Histogram-Report, stellt ein Histogramm und die dazugehörigen Statistiken, in einem Diagramm dar. Für die Darstellung wird theoretisch nur das Aggregations-Kriterium benötigt, das Ergebnis wäre dabei weniger von Nutzen.

Im praktischen Einsatz wird jedoch ein spezielles Dimensions-Modell verwendet:

$$(Width \otimes All_{Spaces}) \sim DW$$

Hier wird über die Kombination aus den Design-Daten, Breite der Struktur sowie Abstand zu den benachbarten Strukturen, eine Aggregation auf die gemessene Abweichung der Breite durchgeführt. Das Resultat zeigt den Einfluss von Abständen auf Breiten.

Abbildung 6 zeigt für das obengenannte Modell (Messung Sample8_CDU) den Einfluss von Abständen (Space: Blau 0-100, Orange 100-1000) und der Breite (Width: 80) auf die gemessene Abweichung. Wie erwartet ist zu erkennen, dass bei kleinen Abständen die Breite des gesamten Histogramms und somit die Standardabweichung der Statistik größer ist, wie das bei größeren Abständen. Schwerwiegender für die Produktion ist jedoch der Mittelwert. Dieser sagt aus, wie weit der gemessene Abstand vom erwartetem entfernt ist.

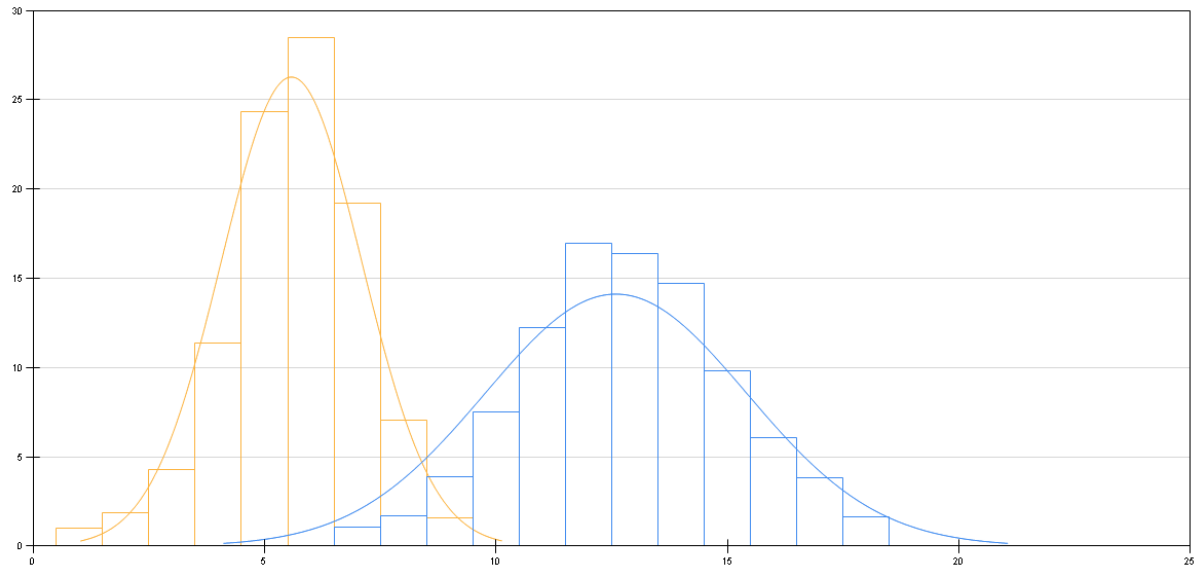

Abbildung 6 – Histogram Plot

Template-Eigenschaften

Um die Analyse an diesem Report zu erleichtern, müssen zusätzliche Optionen eingeführt werden. Die Y-Achse muss konfigurierbar sein, hierzu werden dem Benutzer folgende Optionen gegeben:

- Anzahl

  Es wird die Anzahl der Treffer pro Säule dargestellt.
- Logarithmus

  Die Anzahl der Treffer kann zum Mittelwert, abhängig von der Abweichung, sehr stark zunehmen und somit wird die Y-Achse sehr groß. Dies führt dazu, dass die kleinen Werte nicht mehr dargestellt werden und hat Auswirkung auf andere Histogramme. Um diesem entgegen zu wirken, basiert die Y-Achse nicht auf den Werten (Anzahl der Treffer), sondern auf dem Logarithmus. (siehe Abbildung 6)
- Prozent

  Bei zwei oder mehreren Histogrammen mit sehr unterschiedlichen Höhen (Trefferanzahl), kann es passieren, dass kleine Histogramme zu Flach gezeichnet werden und für eine

Auswertung nicht genutzt werden können. Durch den Einsatz einer Y-Achse mit einer prozentualen Darstellung kann dem entgegengewirkt werden. Dabei werden die Treffer, innerhalb einer Säule, relativ zur Gesamtsumme aller Treffer im Histogramm dargestellt. Die Summe aller Klassen eines Histogramms ist dabei immer 1 (=100%). Dies gilt für alle Histogramme im Diagramm, wie in Abbildung 7 dargestellt.

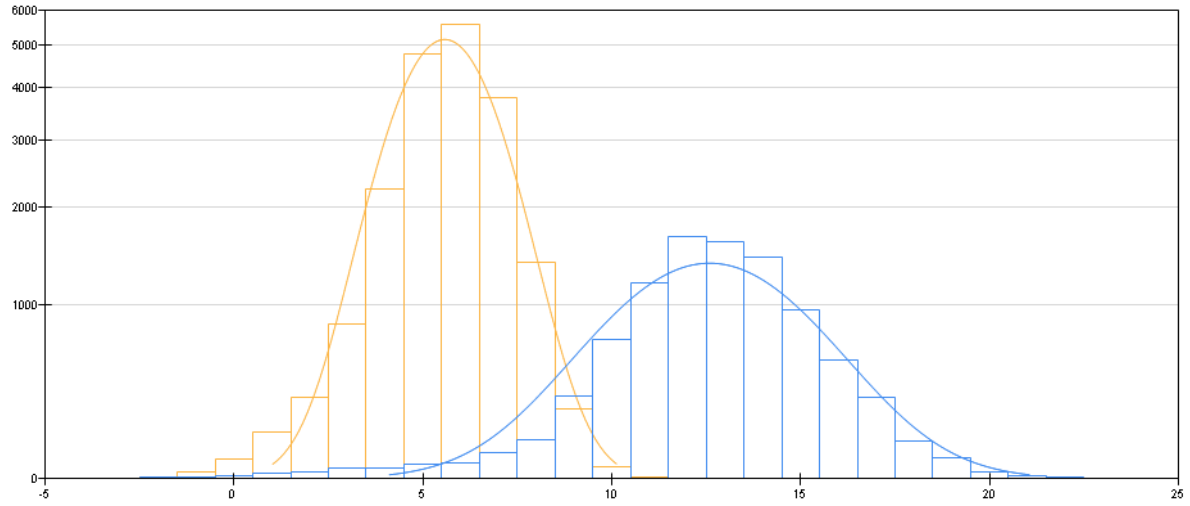

Abbildung 7 – Histogram Plot

Der Histogram-Report kann ebenfalls dazu verwendet werden, um die Normal-Verteilung, die ein Bestandteil der Aggregation ist, abzubilden. In Kombination mit Histogrammen kann eine Aussage getroffen werden, ob die Säulen der Normal-Verteilung folgen. Die Darstellung der drei Optionen (Histogramm, Statistik, Histogramm und Statistik) wird ebenfalls konfigurierbar.

4.2.3 Probabilty-Report

Ergänzend zum Histogram-Report wird der Probability-Report hinzugefügt, hierbei handelt es sich um einen QQ-Plot[4]. Mit diesem ist es möglich, einen Soll-Ist-Vergleich über die Normalverteilung der Histogramme durchzuführen. Es erfolgt ein Vergleich der Flächen der Statistiken untereinander. Die X-Achse zeigt dabei den gemessenen (empirischen) Quantil und die Y-Achse den theoretischen Quantil[5] an. Über die dabei entstandene Punktewolke kann eine Gerade gezogen werden, die mit Hilfe des Verfahrens der kleinsten Quadrate ermittelt wird und die Bezeichnung Regressionsgerade trägt. Für den Fall, dass alle Punkte auf der Geraden liegen, unterliegen die Histogramme einer Normalverteilung.

[4] Quantil-Quantil-Plot
[5] In der Anwendung als Sigma

Für die Darstellung müssen die gemessenen Werte eingetragen werden. Da es sich in hier um aggregierte Werte handelt, müssen sie aus den Histogramm-Säulen ermittelt werden.

Basierend auf dem Beispiel aus dem Histogram-Report Abbildung 7 wird im Abbildung 8 der entsprechende QQ-Plot dargestellt. Dabei ist zu erkennen, dass die Punkte sehr nahe an der Geraden liegen und somit einer Normalverteilung ähneln. Vergleicht man in diesem Fall mit den Histogrammen in Abbildung 7 , so wird die Aussage betätigt.

Template-Eigenschaften

Dieser Report besitzt keine Konfigurationsmöglichkeiten

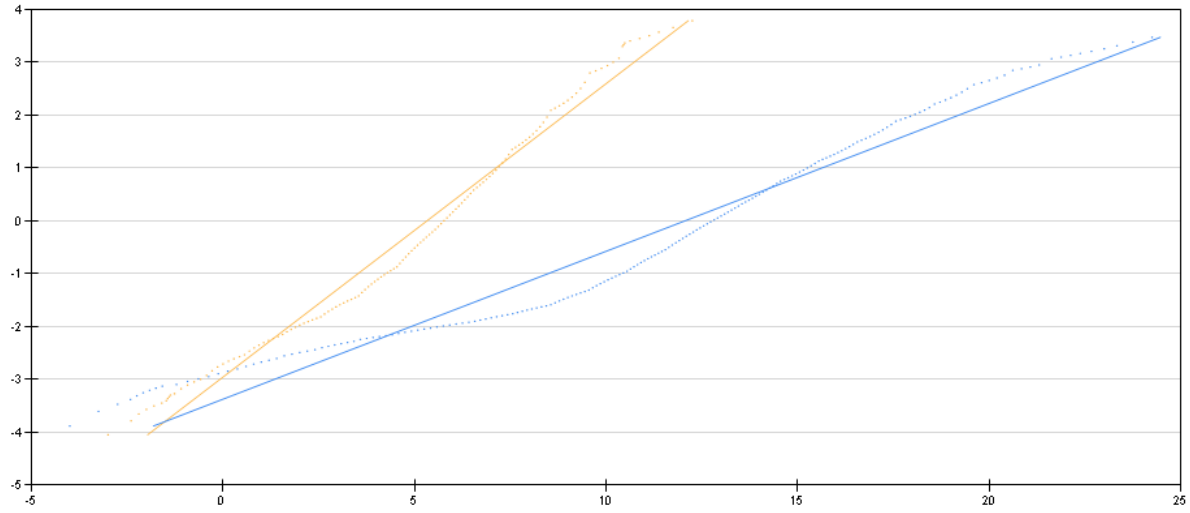

Abbildung 8 – Probability Plot

4.2.4 Statistic-Report

Eine Darstellung von Statistiken in Tabellen wird mit Hilfe des Statistic-Reports realisiert. Anders wie im Histogram-Report wird hier die Statistik nicht als Glockenkurve dargestellt, sondern mit statistischen Kennzahlen in Spalten. Es bietet dem Benutzer die Möglichkeit, nach einer oder mehreren Spalten zu sortieren und gruppieren, um eine besser Übersicht über die Statistik zu erhalten. Des Weiteren besteht die Möglichkeit, mehr als zwei Dimensionen auf einmal zu darzustellen.

Abbildung 9 zeigt den Statistic-Report über das folgende Dimensions-Model für die Messdatei Sample8_CDU:

$$(Width \otimes Space1 \otimes Space2) \sim DW$$

Hier wird eine Kombination aus den Werten, Breite und den beiden Abständen zu dem aggregierten Wert, der gemessenen Breiten-Abweichung, abgebildet. An diesem Beispiel kann man sehen, dass die erste Kombination eine große Anzahl an Treffern[6] besitzt. Für die Statistik bedeutet es, dass sie aus vielen Werten entstanden ist und eine glaubwürdigere Aussage gibt. Die letzte Zeile zeigt einen sehr niedrigen Durchschnittswert[7] und die Standardabweichung[8] ist aufgrund der wenigen Treffern als unglaubwürdig zu betrachten. Diese Informationen würde im Histogram-Report nicht so stark auffallen bzw. untergehen.

Width	Space1	Space2	Ave	Sigma	Min	Max	Range	Count
78	136	136	5,76	1,32	0,1	12	11,9	18304
60	136	136	7,53	1,1	3,4	12,6	9,2	18304
70	145	140	7,67	3,26	-0,3	22,5	22,8	10560
70	140	145	7,77	3,22	-0,3	20,5	20,8	10560
70	995	140	3,32	1,29	1,5	4,5	3	4
70	140	995	4,35	0,49	4	4,7	0,7	2
60	140	995	0,8	0,57	0,4	1,2	0,8	2

Abbildung 9 – Statistic-Table

Template-Eigenschaften

Es werden für diesen Report keine Konfigurationsmöglichkeiten geboten, da alle Informationen von entscheidender Bedeutung für eine Analyse sind.

4.2.5 Simple-Report

Der Simple-Report stellt ein einfaches Linien-Diagramm dar. Die X-Achse repräsentiert eine Dimension und die erste Y-Achse stellt das Quantile über den Aggregierten Wert, sowie die Standardabweichung und Anzahl der Treffer dar. Während die zweite Achse die Anzahl der Messungen darstellt.

Die Standardabweichung und Trefferanzahl wird dabei aus der Statistik gewonnen. Die Quantil-Ermittlung darf nur aus den Histogrammen gewonnen werden. Eine Zuhilfenahme der Statistik, z.B.: Durchschnitt für 50%, verfälscht die Darstellung.

Template Eigenschaften:

Konfigurierbare Quantile, erlauben es den Benutzer eine feinere Abstimmung in der Visualisierung zu ermöglichen.

[6] Count
[7] Ave
[8] Sigma

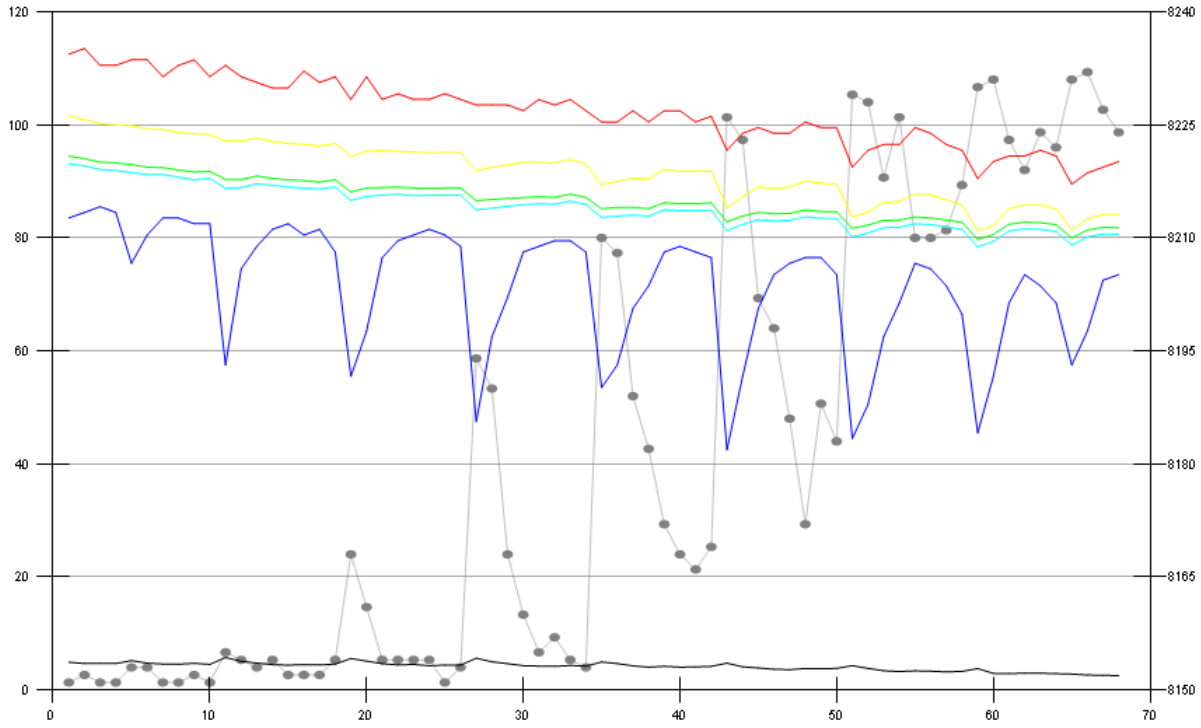

Abbildung 10 - Simple Plot

4.2.6 CD-Uniformity-Report

Der CD-Uniformity-Report stellt die Statistik einer Aggregation auf einem Wafer dar. Anderes wie bei den vorherigen Reports besitzt er unterschiedliche Visualisierungformen: 2D-, 3D-Chart sowie Daten-Tabelle.

Eine Besonderheit des Reports ist das sehr komplexe Modell. Dieses kann nicht mithilfe des DimensionModel modelliert werden. Dies hat zur Folge, dass das DimensioModel eine fest definierte SelectionModelList zurück liefern muss:

$$(Shot_X \otimes Shot_Y) \sim DW$$
$$(Die_X \otimes Die_Y) \sim DW$$
$$(X \otimes Y) \sim DW$$
$$(X \otimes Y \otimes Die_X \otimes Die_Y \otimes Shot_X \otimes Shot_Y) \sim DW$$
$$(Shot_X Shot_Y \otimes Die_X \otimes Die_Y) \sim DW$$
$$(Die_X \otimes Die_Y \otimes X \otimes Y) \sim DW$$
$$(IW_X \otimes IW_Y) \sim DW$$
$$(SMF) \sim DW$$
$$(SMF) \sim IW_X$$
$$(SMF) \sim IW_Y$$

Aufgrund der Komplexität wird dieser Report nicht genauer betrachtet.

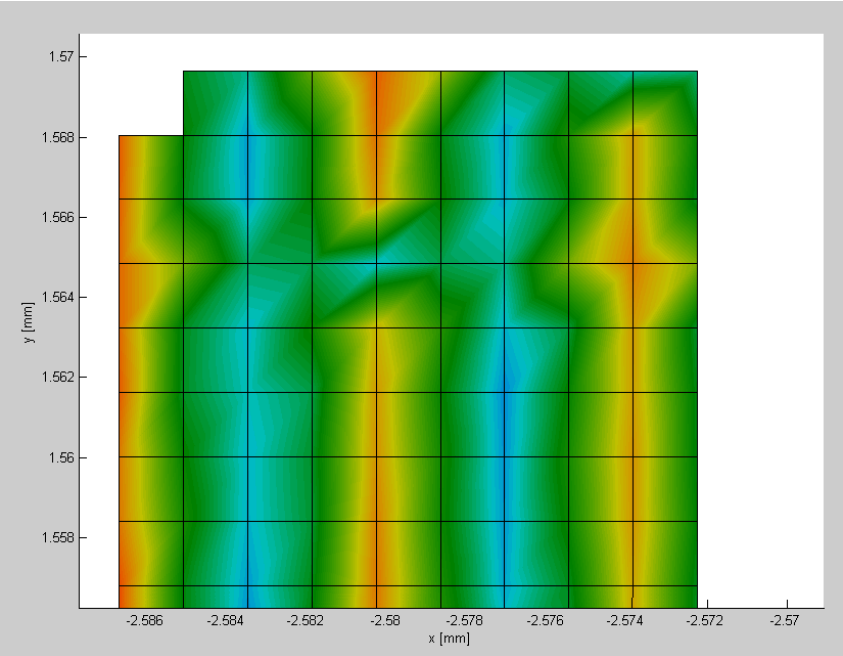

Abbildung 11 – CDU Model

4.3 Client Server Architektur

Betrachtet man die Anforderung von Pendse und Creeth, insbesondere Fast und Shared aus der Client-Seite, so kommt man zur Schlussfolgerung, dass nur eine Client-Server-Architektur zum Einsatz kommen kann, um alle Anforderungen zu erfüllen.

Shared besagt, dass eine Mehrbenutzerumgebung geschaffen werden muss. Dies macht nur in einer Client-Server-Umgebung Sinn.

Fast besagt, dass eine möglichst schnelle Antwortzeit zu gewährleisten ist. Dies kann jedoch nur bedingt bei einer Desktop-Lösung gewehrleistet werden. Da man im Vorfeld nicht weiß, welche Hardware bei Kunden eingesetzt wird. Durch Mindestanforderungen an die Hardware kann dies umgangen werden, aber es sollte nicht von Bedeutung sein, da ein Kunde nur im seltensten Fall bereit, ist seine komplette PC-Landschaft für ein Softwareprodukt aufzustocken.

Leistungstest

Um sich ein Bild über das Antwortverhalten der Aggregation-Engine zu schaffen, muss sie von realistischen Messdaten und Modellen getestet werden. Die Tests unterscheiden sich in der Komplexität der Aggregation-Modellen, sowie der Anzahl der gemessen Punkte und der Dateigröße. Dabei werden die durchschnittliche Prozessor-Auslastung, die Größe der Daten-Cubes sowie der maximalen Arbeitsspeicherverbrauch ausgewertet.

Folgende Messdateien werden für den Test verwendet:

- Measurement1: Sample2_new.csv - 58 MB
- Measurement2: Sample5_new.csv - 790 MB
- Measurement3: Sample1_5Shots_22032010.csv - 12.5 GB

Des Weiteren wird jede Datei mittels der folgenden Modelle aggregiert:

- Model1: $(Width \otimes Space) \sim DW$
- Model2: $\left(Width \otimes All_{Spaces}\right) \sim DW$
- Model3: CD-Uniformity-Modell (siehe 4.7.6)

Für einen aussagekräftigen Leistungstest muss man an dieser Stelle mehr Messdateien und Modelle miteinander vergleichen. Diese 3x3-Kombination reicht hier für eine grobe Übersicht.

Tabelle 3 – Arbeitsspeicherverbrauch

Maximaler Arbeitsspeicherverbrauch [MB]			
	Measurement1	**Measurement2**	**Measurement3**
Model1	113	131	227
Model2	120	420	260
Model3	142	2012	446

Aus Tabelle 3 kann man den maximalen Speicherverbrauch bei der Aggregation entnehmen. Am Beispiel Model3-Measurement2 kann man einen sehr hohen Speicherverbrauch erkennen, daraus folgt, dass die Aggregation-Engine nur in einer 64-Bit Umgebung funktionieren kann.

Tabelle 4 – CPU-Zeit

Durchschnittliche CPU-Zeit [%]			
	Measurement1	**Measurement2**	**Measurement3**
Model1	18	70	78
Model2	34	74	82
Model3	60	63	81

Bei der in Tabelle 4 dargestellten durchschnittlichen, sehr hohen, CPU Auslastung sieht man über die Werte aus der kleinen Messung Measuremtn1 hinweg. Eine solche Auslastung auf einem Client PC haben zur Folge, dass andere Anwendungen wenig CPU-Zeit zur Bearbeitung ihrer Aufgaben erhalten, oder dass die Aggregation-Engine sich diese teilen muss und somit eine längere Bearbeitungszeit zustande kommt.

Beide Auswertungen zeigen, dass die Aggregation-Engine nicht auf einem Client-System ausgeführt werden darf.

4.3.1 Grundlagen

Bei der Client Server Architektur handelt es sich um ein Software-Konzept, womit einzelne Prozesse einer Anwendung auf vernetzte Computer verteilt werden, und somit können voneinander unabhängige Instanzen eine gemeinsame Aufgabe lösen. Hierzu stellt ein Client eine Anforderung/Aufgabe (Request) an einen Server, die entsprechende Dienste (Services) und Daten zur Verfügung stellt. Der Server wiederum bearbeitet die Anforderung und liefert eine entsprechende Antwort (Request) an den Client zurück. Abhängig von der Architektur kann der Server Dienste anderer Instanzen, beispielsweise eines DBMS, in Anspruch nehmen. [Ben04, 27-29]

Die Anfragen der Clients die am Server ausgeführt werden, können auf zwei unterschiedlichen Arten durchgeführt werden. [Ben04, 31]

- Synchrone (blockierend)
  Bei der synchronen Anfrage wartet der Client auf die Antwort des Servers. Dieses Verfahren gilt als ineffizient, da die Client-Anwendung für die ganze Zeit, von der Anfrage bis zu der Antwort, geblockt wird.

- Asynchrone (nicht blockierend)
  Anders wie bei der synchronen Anfrage wartet der Client nicht auf die Antwort des Servers, sondern kann seine Arbeit fortsetzen. Dies wird dadurch ermöglicht, dass der Client seine Anfrage über einen anderen Prozess ausführt. Mit einer Callback-Methode, bei der die Antwort des Servers ausgeführt wird, kommt das Clientseitige Ereignis zum Einsatz. Dieses Verfahren benötigt zusätzliche Kontrollmechanismen, da die Antworten des Servers bei mehrfachem Aufruf nicht in erwarteter Reichenfolge auftritt.

Bei der Client Server Architektur kann der Server auch die Rolle eines Proxy Servers übernehmen. Dabei arbeitet der Server als Stellvertreter zwischen Client und dem ihm bekannten Servern und vermittelt die Antworten zwischen den Systemen. Diese Systeme sollen in der Lage sein, eigenständig die richtigen Kommunikationspartner zu wählen und sollen ebenfalls die Antworten Cachen. [Ben04, 69]

Folgende Vorteile können durch die Client/Server Architektur gewonnen werden.

- zentralisierte Ressourcen

  Ressourcen können durch die Clients gemeinsam genutzt werden. Dabei ist nicht nur die Datenhaltung, wie beispielsweise in Form einer zentralisierten Datenbank, sondern auch die Rechenleistung einer oder mehreren Server gemeint. Die Zentralisierung erlaubt es ebenfalls Updates einer Ressource durchzuführen ohne Änderungen am Quellcode des Clients vorzunehmen.

- Spezialisierung

  Jeder Server kann in einem Client/Server-System für seinen Anwendungszweck optimal ausgerüstet werden.

- Skalierbarkeit

  In diesem Kontext versteht man unter Skalierbarkeit die Möglichkeit einer dynamischen Anpassung des Systems auf Änderungen, hinsichtlich des Ressourcenbedarfs. Eine Anpassung kann das hinzufügen oder entfernen zusätzlicher Server-Ressourcen bedeuten [Mas05, 15].

- Administration

  Die Administration kann zentralisiert vorgenommen werden, hierzu zählt neben der Benutzerverwaltung auch die Ressourcen-Zuteilung, die zentral durchgeführt werden kann.

Nachteile:

- Höhere Kosten

  Durch den Einsatz von zusätzlichen Servern steigen die Kosten für die Anschaffung, den Betrieb und Wartungsarbeiten.

- Single Point of Failure

  Unter einem SPoF (einzelne Stelle des Scheiterns) versteht man eine Komponente in einem System, welche beim Ausfall einen Ausfall des kompletten Systems nach sich zieht [Kev01, 31]. Eine Abhilfe schafft dabei nur ein redundantes Netz. Alle Server müssen im Netz mehrfach vorhanden sein und sich gegenseitig abgleichen.

4.3.2 Kommunkationsplattformen unter .NET

Ein essentieller Bestandteil einer jeden Server Client Lösung ist die Möglichkeit der Interprozesskommunikation zwischen Objekten, die auf unterschiedlichen Prozessen, sich auf einen Lokalen oder entfernten Rechner befinden können.

Das .NET-Framework stellt eine Vielzahl an Low-Level IPC-Mechanismen wie TCP/UDP, Message Queue und Named Pipes dem Entwickler zur Verfügung. Die Mechanismen sind für den Einsatz in einer Client/Server-Umgebung eher bedingt geeignet, da bei der komplexen Implementierung die Fehlerwahrscheinlichkeit sehr hoch und eine Wartung sehr schwierig ist.

Eine Abhilfe sollen die beiden Frameworks .NET Remoting, sowie die neuere Windows Communication Foundation (WCF), schaffen.

Ein Vergleich der beiden Ansätze wird an dieser Stelle nicht stattfinden, da eine Integration in die DFMSim Software Architektur eine .NET-Remoting Anbindung voraussetzt.

4.4 Antwortzeiten

Ausgehend vom ersten Punkt (Fast) der FASMI-Regeln, ist eine möglichst schnelle Antwortzeit auf Anfragen des Benutzers zu gewährleisten. Die in der Regel vorgesehenen Zeiten sind jedoch sehr stark von den Messdaten und den Aggregation-Model anhängig.

Leistungstest

Der Leistungstest erfolgt für eine Antwortzeitanalyse, mit Hilfe der in 4.2 beschriebenen Messdaten und Modelle sowie das dazu erweiterte Testtool, dass eine Aggregationszeit und resultierende Dateigröße ausgegeben wird.

Tabelle 5 - Aggregationzeit

Aggregationszeit [min]			
	Measurement1	**Measurement2**	**Measurement3**
Model1	0.1	0.7	9.5
Model2	0.1	1.2	13.7
Model3	0.2	2.7	33.5

Aus Tabelle 5 kann man erkennen, dass die Bearbeitungszeit von der Dateigrösse sowie den Dimensionsmodell abhängig ist. Dies ist mit der Arbeitsweise der Aggregation-Engine zu erklären. Die Daten werden bei der Bearbeitung sequenziell, also Zeile für Zeile, eingelesen. Die so

ausgelesenen Zeilen werden den Prozess der Aggregation unterzogen, dieser direkt von den SelectionModelList abhängig ist.

Tabelle 6 - Cubegröße

Cubegröße [MB]			
	Measurement1	**Measurement2**	**Measurement3**
Model1	1.1	6.2	0.1
Model2	7.9	44.8	0.6
Model3	2.5	363.9	17.9

Die Dateigröße der aggregierten Daten ist nicht direkt von der Messdateigröße abhängig. Tabelle 6 zeigt, dass die Abhängigkeit im Modell und somit in der SelectionModelList zu finden ist. Bei der Aggregation können Spalten gruppiert werden und auf einer Spalte wird eine Aggregation mit der Histogram-Klasse vorgenommen, dabei kommt es zu einer Datenreduzierung. Somit ist die resultierende Datengröße vom, für die Aggregation relevanten, Inhalt der Messdatei anhängig. Am Beispiel Model1-Measurement3 kann man sehen, dass bei der Messung entweder eine Width- oder ein Space-Gruppe untersucht worden ist.

Die im Allgemeinen langen Aggregationszeiten haben zur Folge, dass eine Echtzeitanalyse nicht möglich ist. Um dieser Sachlage entgegenzuwirken muss eine Lösung entwickelt werden, welche die Zugriffszeiten, auf die vom Benutzer geforderten Resultate, auf ein Minimum reduziert.

4.4.1 Serialisierung

Bei der Serialisierung wird der Zustand eines Objektes in einem sequenziellen Datenstrom von Bytes gespeichert. Dabei wird der Datenstrom im Speicher, in einen nicht flüchtigen Zustand, in einer Datei oder Datenbank abgelegt. Der umgekehrte Vorgang, das Umwandeln des Datenstroms in ein Objekt, wird als Deserialiserung bezeichnet [WWW02].

Das Speichern der verdichteten Messdaten in einen persistenten Zustand stellt die einfachste aber auch die effizienteste Form, die Antwortzeiten zu beschleunigen, dar. Da an den Messdateien keine Änderungen von außen vorgenommen werden, können die Aggregate als absolut betrachtet werden. Eine Konsistenzkontrolle ist somit, bezogen auf den Inhalt, nicht notwendig.

Die Serialisierung der Aggregate in einer Datenbank, kann an dieser Stelle nicht vorgenommen werden. Legt man die Objekte einzeln in Tabellen ab, kann es aufgrund der Indexierung einen sehr großen Overhead für das Datenbanksystem bedeuten.

Das .NET-Framework stellt vier Serialisierung-Verfahren dem Entwickler bereit, drei davon werden im Verlauf vorgestellt [WWW02]:

Binäre- Serialisierung

Der Datenstrom wird binärcodiert gespeichert. Dabei werden alle Members eines Objektes, auch private und schreibgeschützte, mit übertragen. Nach der Deserialiserung werden mehrfach referenzierte Members des Objektes als Referenzen zurückgeführt. Die Binäre-Serialisierung wird im Allgemeinen als die leistungsstärkste angesehen. Dies betrifft die De-/Serialisierung-Geschwindigkeit sowie die Größe des Datenstromes.

XML-Serialisierung

Bei der XML-Serialisierung werden nur öffentliche Members serialisiert. Der Datenstrom wird in einem XML-Stream gespeichert, welcher gegen eine bestimmte XSD validieren kann. In der XML werden referenzierte Members nicht als solche gekennzeichnet, somit entstehen bei der Deserialiserung nur Klone. Aus diesem Grund sind auch keine zirkulären Referenzen möglich, da diese zu einer Endlosschleife und somit zu einem Speicherüberlauf führen. XML-Dateien haben den Vorteil, dass sie Plattformunabhängig behandelt werden können.

SOAP- Serialisierung

Bei der SOAP-Serialisierung handelt es sich um eine XML-Serialisierung. Eine Besonderheit dieses Verfahrens ist, dass hier auch private Members gespeichert werden und bei der Deserialiserung Referenzen richtig aufgelöst werden. Es ist zu beachten, dass keine Generic-Collections verwendet werden können.

Vergleich

Das Laden und Speichern der aggregierten Daten ist ein wichtiges Kriterium für schnelle Antwortzeiten. Hierzu wurde eine Test-Anwendung (siehe Anhang) entwickelt, welche die einzelnen Serialiserungsklassen auf Zeiten und Dateigröße testet. Zusätzlich zu den obengenannten Testanwendungen wurden zwei weitere Methoden hinzufügt. DataContract und NetDataContract, beide sind XML-Basierend. Die Tests werden mit einem Struct sowie einer Klasse durchgeführt.

Zwei der Serialisierungsverfahren stehen besonderes hervor. Betrachtet man Abbildung 12 so erkennt man, dass die binäre-De/Serialisierung (BinaryFormatter) wie erwartet am schnellsten ist. Hierbei wurden Structs gespeichert und geladen.

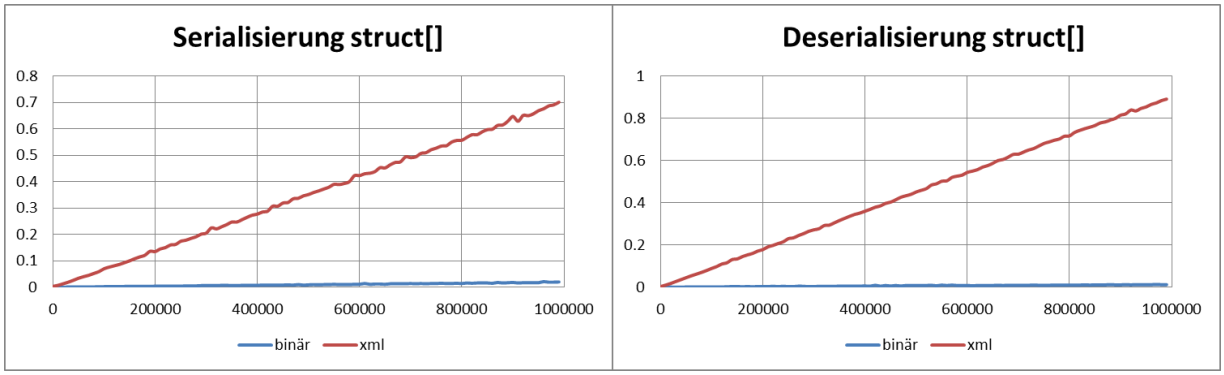

Abbildung 12 – Struct[] Test

Abbildung 13 zeigt für den Fall, dass Klassen gespeichert und geladen werden, ein komplett anderes Bild. Hier ist insbesondere bei der Deserialierung der XmlSerializer im Vergleich zur BinaryFormatter schneller. Die Binäre-Deserialiesierung steigt im vorliegenden Fall leicht quadratisch an.

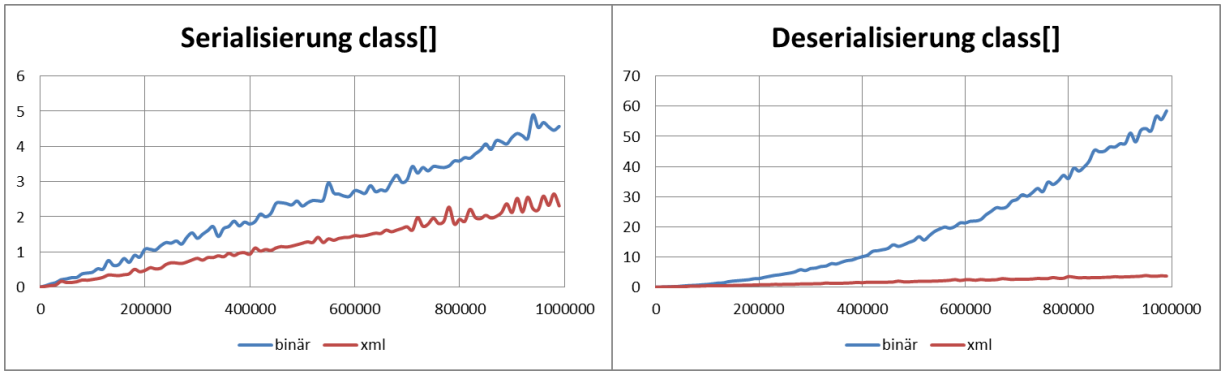

Abbildung 13 – Class[] Test

Ein Erklärungsversuch könnte darin liegen, dass bei der Binären-Serialisierung auch nicht sichtbare (private) Members gespeichert werden. Im Vorliegenden Fall wurden keine privaten Members definiert, somit ist diese Erklärung hinfällig. Viel eher ist eine Erklärung darin zu finden, dass der BinaryFormatter im Gegensatz zum XMLSerializer einer Refrenzenwiederherstellung gewährleistet. Dies ist der einzige Unterschied zwischen den beiden Test-Szenarien.

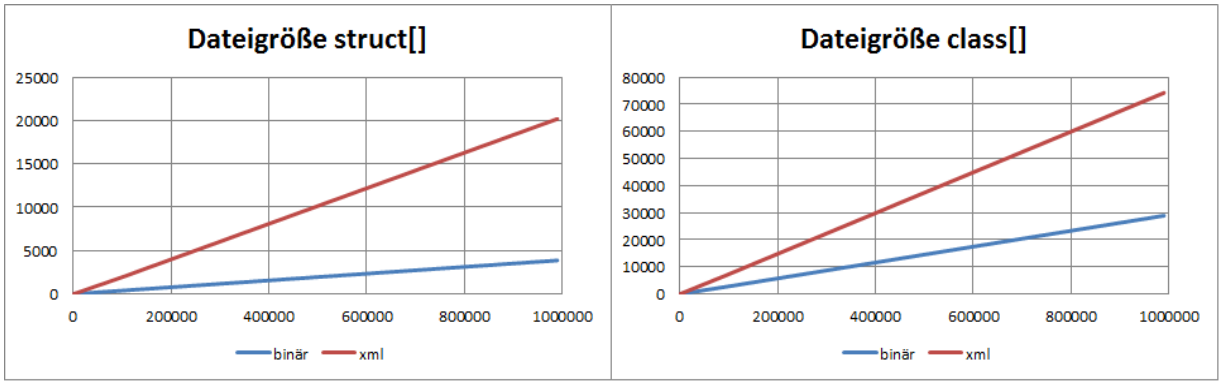

Abbildung 14 - Dateigröße

Die resultierende Dateigröße des XMLSerializers ist in beiden Fällen groß (Abbildung 14). Für die Analyse Software spielt die Dateigröße keine relevante Rolle. Interessanter ist die Ladezeit, da sie ein Kriterium für eine performate Antwortzeit ist. Aufgrund der Ergebnisse wird für eine Serialisierung der ClusterData der XMLSerializer verwendet.

4.4.2 Zugriff auf aggregierte Daten

Wie aus Tabelle 6 erkennbar ist, kann trotz einer Aggregation die resultierende Datei sehr groß werden. Es folgt eine Übertragung der kompletten Datei vom Server auf den Client, die nur bedingt anwendbar ist.

Um die Daten von Server zum Client zu übertragen, muss ein Kommunikationskonzept verwendet werden, das in der Lage ist Daten über das Netzwerk zu transportieren. Hierzu findet man in den .NET Framework unterschiedliche Ansätze (4.3.2).

All diese Konzepte haben den unausweichlichen Nachteil, dass die Daten zur Übertragung vor dem Senden serialisiert (T_{SSer}) und am Empfangen deserialisert (T_{RDes}) werden müssen. Die aggregierten Messdaten auf dem Server, müssen vor einer Übertragung aus ihrem gespeicherten Zustand, durch eine Deserialiserung (T_{SDes}) in den Speicher geladen werden. Das Senden dauert eine gewisse Zeit (T_{send}) und dazu folgt die Reaktionzeit des Servers (T_{Latenz}), welche von der Auslastung abhängig ist.

Insgesamt kann die Antwortzeit wie folgt ermittelt werden:

$$T_{Response} = T_{Latenz} + T_{SDes} + T_{SSer} + T_{Send} + T_{RDes}$$

Die Übertragungszeit wäre an dieser Stelle für den Benutzer zu hoch.

4.4.2.1 Seitenweiser Zugriff

Bei einem Zugriff auf die Analysen ist der Benutzer nicht am kompletten Inhalt interessiert, sondern an einer Seite die den gewünschten Report darstellt. Somit ist eine Übertragung eines kompletten Datenwürfels überflüssig. Mit diesem Verfahren könnten die Zeiten für T_{SSer} ,T_{Send}, T_{RDes} stark reduziert werden.

An dieser Stelle muss erst eine Möglichkeit geschaffen werden, eine Liste mit Seitenbeschreibungen vergleichbar mit einem Inhaltsverzeichnis, an den Client zurückzuschicken. Zusätzlich muss eine Möglichkeit geschaffen werden, bei der man mit Hilfe eines Indexes auf die Histogramme innerhalb der Seite zuzugreifen kann. Der Aufruf ist stark von den Reporttypen und deren Konfiguration anhängig.

Zugriff über Iterator

Über den aggregierten Daten kann ein Iterator abgerufen werden. Hierzu wird eine Beschreibung benötigt, bei der die Dimensionen innerhalb des Dimensionmodells für die Seitenbeschreibung zuständig sind, und zugeteilt, wie viele Series der Report erhalten darf. Die AxesDescriptions beinhalten die Informationen, die für eine sortierte Seitenbeschreibung nötigt sind. Neben der Achsenbeschreibung werden hier auch die zugehörigen Histogramme zurückgeliefert. Sie werden für eine Seitenbeschreibung nicht benötigt und dürfen auch nicht an die aufrufende Instanz gesendet werden, da sie die Histogramme beinhalten.

Der Zugriff auf die Histogramme einer Seite, erfolgt auf gleichem Weg, dabei wird der Index der Seite angegeben.

Seitenübersicht über ClusterData

Über einem direkten Zugriff auf die das ClusterData[]kann ebenfalls eine Seitenbeschreibung generiert werden. Hierzu muss durch alle Cubes, und innerhalb dieser alle Scales und ScaleItems, iteriert werden.

Dieses Verfahren wird nur bei dem CdUniformity-Report eingesetzt, da der Zugriff auf diesem Weg nur schwer, aufgrund der unterschiedlichen SelectionModels, des Iterator zu lösen ist.

4.4.2.2 Caching

Eine einfache Möglichkeit $T_{Response}$ zu verkleinern ist die Reduzierung der T_{SDes} Zugriffe. Ein Ansatz ist, die geladenen Daten im Hauptspeicher zu behalten und nicht bei jedem Zugriff zu entfernen. Dies hat den Nachteil, dass Aufgrund der Datengröße, der Speicher vom Server nach kürzester Zeit verbraucht ist und somit auch T_{Latenz} ansteigt. Dieses Vorgehen ist im Ansatz richtig, bedarf jedoch

einer Funktionalität die dafür sorgt, dass der Speicher des Servers nicht zu so stark ausgelastet ist. Hierzu sollte eine Caching Lösung in Betracht gezogen werden.

Beim Cache (deutsch Pufferspeicher) handelt es sich um ein Verfahren, Daten die schnell verfügbar sein sollen, in einem schnelleren Speicher zwischen zu lagern.

Im vorliegenden Fall ist die Serialisierung der aggregierten Daten die einfachste Form des Cachings. Dieses Verfahren ist zum einen einfach in der Implementierung und zum anderen, im Vergleich zur Deserialisierungszeit und Aggregierungszeit, sehr effizient. Die Probleme die dieses Verfahren mit sich bringt wurden in 4.4.2 diskutiert.

Hierzu werden zwei Ansätze vorgestellt

SimpleCache

Hierbei handelt es sich um ein Caching-System welchem eine bestimmte Hauptspeichergröße zugewiesen worden ist. Das System muss somit in der Lage sein den eigenen Speicher zu beobachten. Desweiterem muss eine Tabelle geführt werden, in der die zwischengespeicherten Daten und den dazugehörigen letzten Zugriffszeiten enthalten sind. Kommen Anfragen an das Caching-System, so muss es entsprechend auf die Daten angepasst werden. (siehe Abbildung 15).

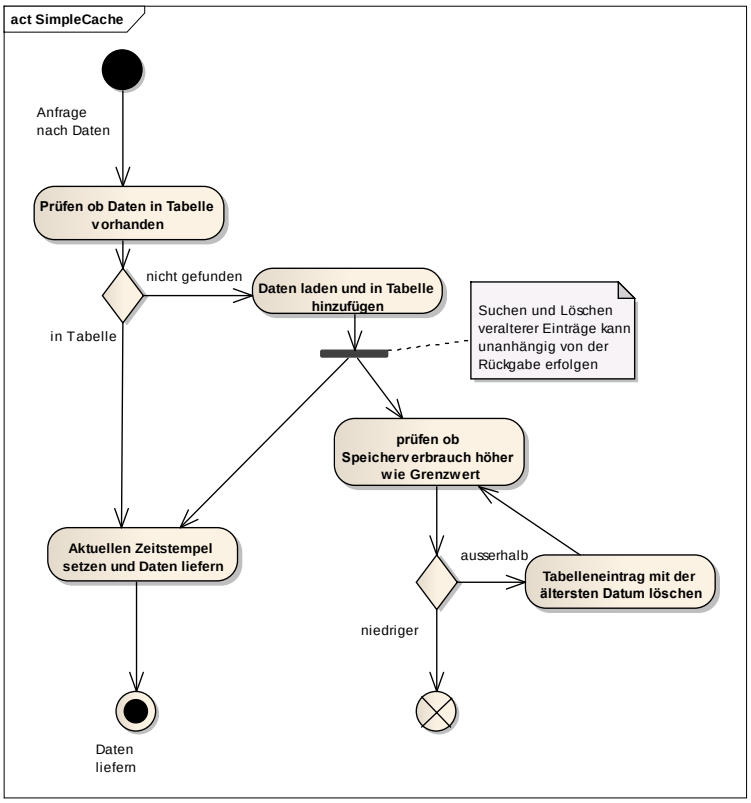

Abbildung 15 – SimpleCache

PageCache

Der PageCache vereinigt den in 4.4.2.1 vorgestellten Seiten-Zugriff mit den SimpleCache. Hierbei wird die Tabelle aus dem SimpleCache durch eine Histogramm-Klasse erweitert. Dies geschieht mit dem Ziel, eine Übersicht über Zugriffe auf bestimmte Seiten zu führen. Die daraus entnommenen Informationen sollen dazu dienen, dass der aggregierte Cube nicht aus der Tabelle entfernt wird, sondern dass auf diesem eine oder mehrere Transformationen durchgeführt werden, welche die meistgesuchten Daten beinhalten.

4.5 Automatisierte Aggregation

Aus internen und Kundengesprächen hat sich herausgestellt, dass viele Analysen dem gleichen Muster, also Dimensions-Modell entsprechen. Das bedeutet, dass es Standard-Modelle für bestimmte Messdaten gibt, die mit der Kontext-Informationen ermittelt werden können. Beispielsweise ist ein Standard Modell für den Histogram Report ein $(Width \otimes Space) \sim DW$.

Somit muss innerhalb der Software die Möglichkeit geboten werden, durch einen Dienst eine automatisierte Aggregation auf der Basis, durch eine vom Benutzer definierten Regeln, zu ermöglichen.

Es ist zu bedenken dass die Automatisierung zu bestimmten Zeitpunkten oder Ereignissen ausgeführt werden kann, folgende Auslöser sind dabei möglich:

- Dateneingang
  Eine neue Messdatei befindet sich im Input-Verzeichnis
- Zeitgesteuerte
  In bestimmten Intervallen oder zu bestimmten Zeiten, zum Beispiel an Tagen, an dennen keine oder wenig Auslastung stattfindet (Freitag oder Sonntag)
- Server gesteuert
  Die System Auslastung an der Aggregation-Engine ist niedrig

Der letzte Punkt zeigt aber auch ein nicht zu ignorierendes Problem. Bei einer schlecht oder falsch konfigurierten Automatisierung, kann der Server unnötig zu einem falschen Zeitpunkt belastet werden. Dies hätte zur Folge, dass die Arbeit eines Benutzers am System unnötig verzögert wird. Aus diesem Grund muss auch an hier eine Priorisierung stattfinden.

Mit der automatischen Aggregation ist es ebenfalls möglich, Antwortzeiten des Servers bei Benutzeranfragen zu verkürzen. Da die entstandenen Aggregate für Benutzerdefinierte Reports weiterverwendet werden können.

4.6 Einsatz von Datenbanken

Datenbanken sind für die Speicherung von aggregierten Daten nicht geeignet, da sie sehr komplexe und große Datenstrukturen beinhalten. Für die Speicherung von Projekten, Reports und Kontextinformationen eignen sich Datenbanken sehr gut.

Der Einsatz einer Datenbank bringt einer Reihe an Vorteilen mit:

- Effizienter Datenzugriff
  Ein modernes DBMS bietet Mittel zur effizienten Datenspeicherung und Wiederherstellung. Dazu zählen unter anderem Indexierungsverfahren zur schnelleren Datenselektion.

- Transaktion und Mehrbenutzerfähigkeit
  Zusammenhängende Operationen können in Transaktionen zusammengefasst werden und atomar als Ganzes oder gar nicht ausgeführt werden. Somit können keine konkurrierenden Schreibzugriffen entsteht.

- Datenzugriffkontrolle
  Unter anderem kann, durch Benutzerrollen und Rechte, der Zugriff auf Daten kontrolliert und gesteuert werden.

4.6.1 ADO.NET

Das .NET Framework bietet, ab Version 3.0, abhängig von der eingesetzten Datenbank, zwei Zugriffsverfahren an. Das einfachste und bekannteste ist ADO.NET.

Bei ADO.NET handelt es sich um eine Klassen-Bibliothek, die den Zugriff auf relationale Datenbanken bereitstellt. ADO.NET ist nicht nur an die DBMS der Firma Microsoft gebunden, es finden sich unterschiedliche Data Provider, die jedoch vom Herstellen selbst gewartet werden müssen. Dazu zählen unteranderem Oracle, MySQL und Postgres. Die Kommunikation zwischen der Datenbank und dem Datenprovider erfolgt über die Programmiersprache SQL.

Zugriff durch O/R-Mapper

Um objekt-orientierte Daten in einer relationalen Datenbank zu speichern oder diese wiederherzustellen, muss der Entwickler eine Abbildung zwischen dem Objekt und der Tabellen und schaffen. Beide Systeme besitzen jedoch unterschiedliche Arbeitsweisen und Strukturen. Hierbei spricht man vom „impedance mismatch“ bzw. „object-relational impedance mismatch“, was so viel bedeutet wie Objekt-relationale Unverträglichkeit [Kle10, 4]. Es tritt auf, wenn beispielsweise eine

Vererbungshierarchie von Klassen auf der Datenbank abgebildet werden soll. Hierbei wird eine Datenbank mehrere Tabellen benötigen. In der OOP wird das durch ein Objekt abgebildet.

Bei einen O/R-Mapper (object/relational mapper) handelt es sich um ein Framework, dass die Lücke zwischen der OOP und einer Datenbank schließt. Dazu wird mit Unterstützung des ORM eine Abbildung (engl. Mapping) zwischen Klassen und Tabellen geschaffen.

Im .NET-Framework finden sich zwei O/R Mapper, sowie ORM anderer Anbieter. Der bekannteste Vertreter ist Hibernate (Java) bzw. die .NET-Variante NHibernate. Hierbei handelt es sich um eine sehr verbreitete Open-Source-Lösung, welche insbesondere im Java-Umfeld starken Zuspruch findet. Der Einsatz von Hibernate wird an dieser Stelle nicht betrachtet, da für das Projekt möglichst kein Drittanbieter Tool eingesetzt werden soll.

Bei den beiden Varianten der Firma Microsoft, handelt es sich um „Linq to SQL“ und das „Entity Framework“ auch als „Linq to Entity“ bekannt. Diese besitzen eine sehr gute Integration in der Entwicklungsumgebung Visual Studio (ab Version 2008), wodurch das Mapping und die allgemeine Arbeit komfortabel und effizient geschehen kann. Bei Linq (Language Integrated Query) handelt es sich um eine Syntax-Komponente von C# 3.0 (bzw. VB 9). Diese erlaubt es dem Entwickler, Abfragen auf eine Datenbasis, mit Hilfe einer SQL-Ähnlichen Syntax vorzunehmen.

4.6.2 XML Datenbank

Die Firma DFMSim verwendet bei vielen Projekten, die eine kleine Datenbasis haben oder sich im Entwicklungszustand befinden, eine XML-basierende Datenbank. Dieser Ansatz bringt den Vorteil, dass eine Implementierung unabhängig von Standort erfolgen kann. Da sich die Daten auf dem lokalen System befinden und kein zusätzliches Management System benötigt wird.

Aus diesem Grund wird in diese Phase des Projektes eine XML-Basierte Datenbank verwendet.

4.7 Mehrbenutzerbetrieb

Eine wichtige OLAP-Anforderung ist der Mehrbenutzerbetrieb (Shared). Hierzu zählt nicht nur der kontrollierte Zugriff auf Ressourcen sondern auch auf Dienste. Diese Probleme müssen in der vorliegenden Version der Anwendung nicht betrachtet werden.

4.8 Usability

Bei der Datenaggregation handelt es sich um einen sehr komplexen Vorgang. Die Erstellung der einzelnen Modelle und der Zugriff auf die Ergebnisse erfordert ein sehr fundiertes Wissen über die Arbeitsweise der Engine. Diese muss durch einen einfachen Arbeitsablauf und eine sowohl intuitive als auch mächtige grafische Oberfläche vereinfacht werden.

4.8.1 Project-Reports-Konzept

Um den Benutzer die Arbeit mit der Aggregation-Engine zu erleichtern, wird das Project-Reports-Konzept eingeführt. Dabei werden Reports und die Messdatei durch ein Projekt zu einen gemeinsamen gebunden. Die in der Messdatei befindlichen Kontextinformationen stellen für den Benutzer eine wichtigere Informationsquelle als der reine Dateiname dar. Somit wird die Messdatei durch die Kontextinformationen repräsentiert.

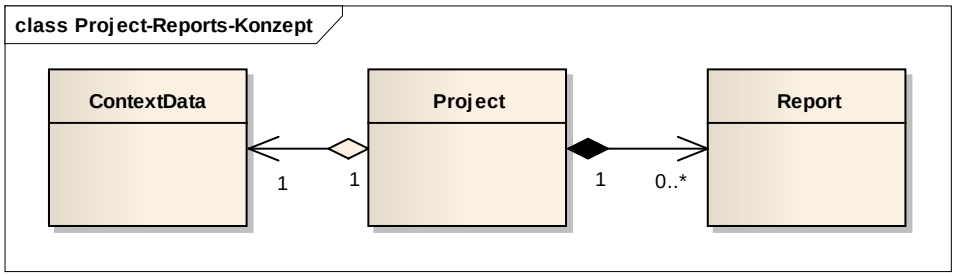

Abbildung 16 – Project-Reports

Diese Aufteilung bringt Struktur in eine Analyse und erleichtert die Navigation zwischen einzelnes Reports.

Neben dem Projects wird es noch ein Template-Project geben, dieses wird jedoch keine Bindung an Kontext-Daten haben und wird für die Automatische Generierung von Projekten zuständig sein. Hierzu wird die in 4.5 beschrieben automatische Aggregation auf Projekten angewendet. Dabei sollen auf bei Auftreten neuer Messdateien, automatisch Projekte und Reports erstellt werden, dies zieht eine Aggregation mit sich.

4.8.2 Benutzeroberfläche

Basierend auf dem Project-Reports-Konzept kann die Oberfläche sehr strukturiert aufgebaut werden. Als Framework für diese wird das DFMSim-AppFrame eingesetzt. Dabei handelt es sich um ein UI-Framework welches einfach um zusätzliche Komponenten erweitert werden kann. Des Weiteren bittet es eine große Funktionsbasis speziell für den Halbleiterbereich, dazu zählen Hilfsmittel zur Arbeit und Darstellung von Wafern.

Für die Anwendung wird ein Plugin entwickelt, welche aus drei Views besteht:

- Reporting Project

  Dieses Plug-In für die Verwaltung von Projekten und Reports zuständig, sowie für die Eigentliche Analyse. Somit ist es die Wichtigste Komponente innerhalb des Plug-Ins.

- Auto Report Setup

  Die Konfiguration der Automatischen Aggregation wird mittels dieses Plug-Ins geschehen. Dies wird in der vorliegenden Version nicht integriert

- Job Monitor

  Der Job Monitor soll für die Überwachung der Aggregation-Engine zuständig sein. Da die Aggregation-Engine derzeit noch keine Möglichkeit bittet, Aggregationen zu stoppen, wird der Job Monitor in einer späteren Version entworfen.

5 Entwurf

5.1 Software-Architektur

Unter dem Begriff Software-Architektur versteht man eine strukturierte Aufteilung einer komplexen Software-Lösung in einzelne und in Beziehung stehender Komponenten (Tabeling 2005, 394).

Eine wichtige Anforderung an die Analysesoftware ist die Trennung des Clients vom Server. Hierbei spricht man von der 2-Tier-Architektur. Die komplette Softwarelösung wird dazu in zwei Schichten aufgeteilt. Die Präsentations- und Anwendungsschicht wird von der Datenhaltungsschicht getrennt [Bal99, 371]. Betrachtet man die in 4.2 beschriebenen Performance-Probleme, so erkennt man, dass dieser Ansatz nicht ausreicht. Die Präsentationsschicht muss von der Anwendungsschicht getrennt werden. Diese drei-schichtige Aufteilung wird als 3-Tier-Architektur bezeichnet. Neben der Trennung ist auch die Kommunikation zwischen den Schichten wichtig. Die Präsentationschicht kommuniziert mit der Anwendungslogik und diese mit der Datenhaltungsschicht. Eine direkte Verbindung zwischen der Benutzerschnittstelle und der Datenhaltung ist nicht erlaubt.

Bei der vorliegenden Lösung wird es sich um eine 3-Schichten-Architektur handeln. Dabei bestehen die Applikationsschicht und die Datenhaltungsschicht aus mehreren Komponenten. In diesem Kapitel werden die einzelnen Schichten und Komponenten in ihrem Aufgabenbereich und Beziehung zueinander beschrieben.

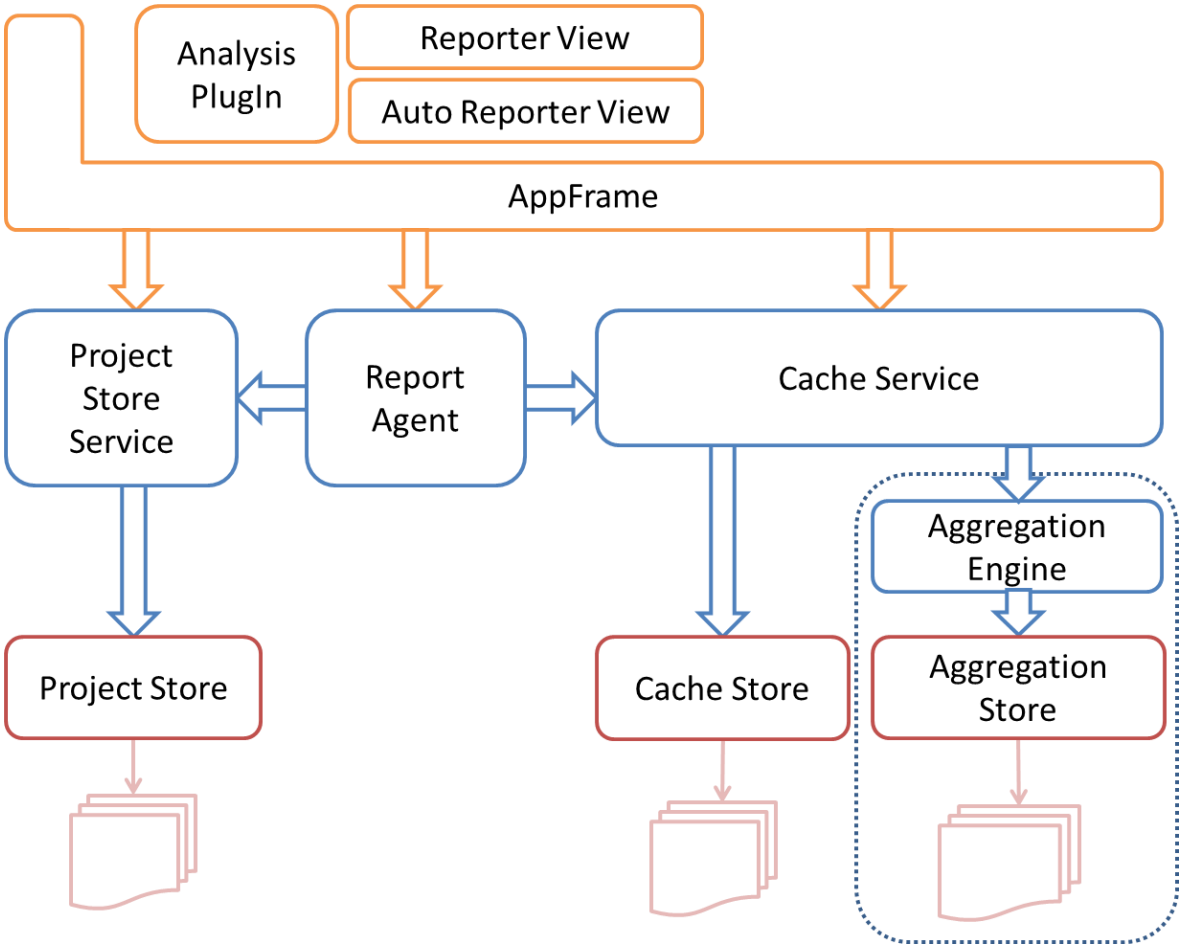

Abbildung 17 - Architektur

5.2 Datenhaltungsschichten

Wie aus der Anforderungsanalyse ersichtlich ist, spielt die Datenhaltung für Analysesoftware eine wichtige Rolle. Die Leistungsfähigkeit wird durch eine Serialisierung der Modelle und Aggregate gesteigert und der Informationsgehalt der Messdaten bleibt erhalten. Die in der Datenhaltungsschicht beschriebenen Komponenten sollen nur für CRUD Operationen zuständig sein. Hierzu zählen das Erstellen (Create), Lesen (Read), Aktualisieren (Update) und Löschen von Entitäten (Objekten) innerhalb eines persistenten Systems. Alle Operationen die den Aufruf ausführen, werden durch die dazugehörigen Engines durchgeführt.

5.2.1 Aggregation Store

Eine Kernaufgabe des Aggregations-Stores ist die Datenhaltung der Messdaten. Durch einen externen Prozess werden die Messdaten in den Store geschrieben bzw. gelöscht. Dieser Vorgang geschieht im Vorliegenden auf dem Datei-System. Änderungen am Inhalt der Messdateien können, nach Aufnahme in den Store, nicht mehr vorgenommen werden.

Da das komplette Konzept der Anwendung auf Kontextdaten beruht, müssen diese zusätzlich in einer XML-Datenbank verwaltet werden. Dies geschieht aus Performance-Gründen, mit dem Ziel ein schnelleres Auffinden der zugehörigen Messdaten zu ermöglichen.

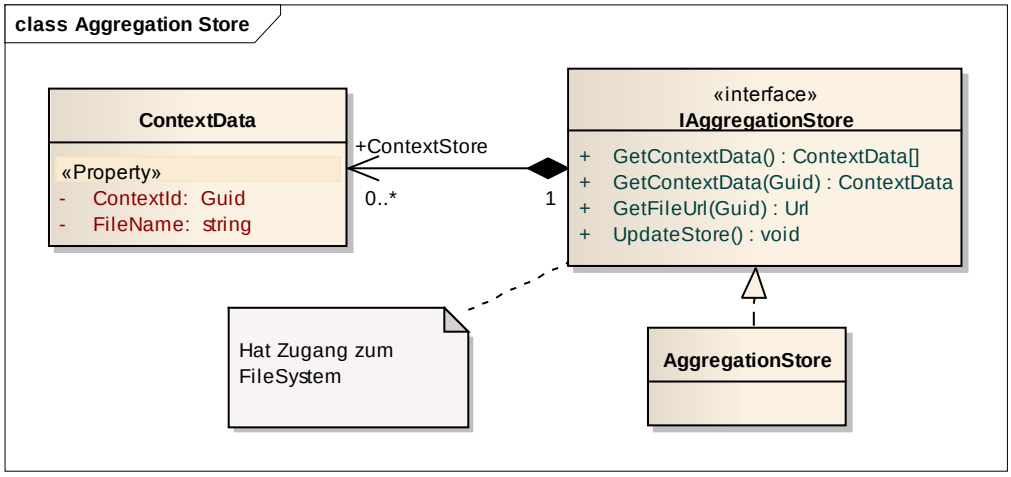

Abbildung 18 – Aggregation Store

Hierzu wurde die ContextData-Klasse um eine Guid erweitert. Hinzu kommt der Dateiname um bei der Projekt Definition einen Anhaltspunkt zu setzen.

Mit der Methode UpdateStore, werden die Kontextinformationen mit den Messdateien im FileSystem abgeglichen. Dies kann dazu führen, dass Kontextdaten gelöscht werden. Diese Methode wird in bestimmten Intervallen ausgeführt.

Diese Komponente ist ein Bestandteil der Aggregations-Engine und wird für den Entwurf nicht genauer betrachtet.

5.2.2 Cache Store

Der Cache Store hat zur Aufgabe bereits aggregierte Messdaten im transienten Zustand zu halten und zu verwalten. Dabei wird das Ziel verfolgt, einen möglichst schnellen Zugriff auf die Daten zu gewährleisten. Es werden dabei nicht nur Cluster-Daten gespeichert, sondern die zugehörige Kontextinformationen sowie das Modell und Transformationen, auf welcher Basis die Aggregation erfolgt ist. Pro ClusterDataAssign werden nur ein SelectionModel und ein ClusterData gespeichert. Dies erlaubt eine Wiederverwendung von aggregierten Daten, unabhängig der SelectionModelList.

Hierbei werden Operationen wie Speichern, Laden, Löschen und Suchen unterstützt. Bearbeitungsoperationen werden nicht bereitgestellt, da die Cubes als unveränderbar (engl. immutable) in den Store abgelegt werden müssen.

Die Datenhaltung der Aggregate erfolgt auf dem Dateisystem und wird mit Hilfe des XmlSerializers realisiert (4.4.1). Das Modell, die Transformationen sowie die zur Messung dazugehörigen Kontextinformationen werden in einer XML-Datenbank abgebildet.

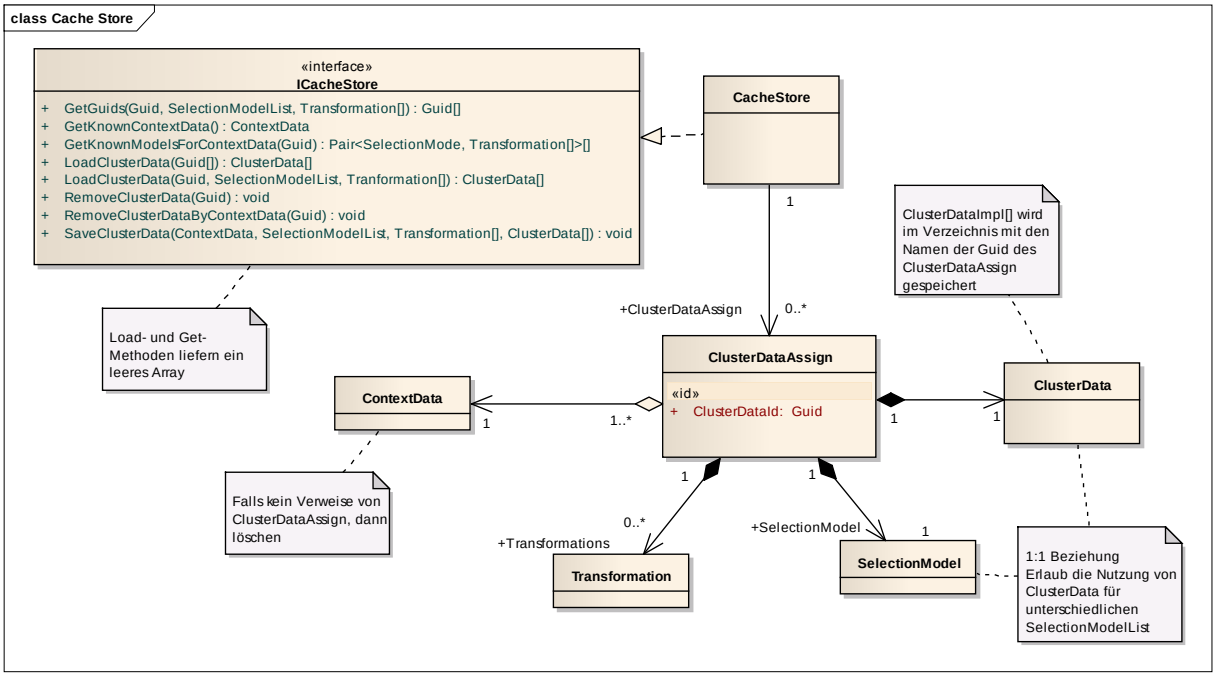

Abbildung 19 –CacheStore

Die ClusterDataId dient nicht nur zur Identifizierung des ClusterDataAssign, sondern auch als Name für das Verzeichnis, in welchem die Aggregierten Messdaten gespeichert wurden. Beide Objekte sind streng voneinander abhängig. Dies hat zufolge, dass beim Löschen eines der beide entfernt wird. Eine vergleichbare Bindung herrscht zu den Kontextdaten. Sie werden gelöscht, wenn keine Verweise vorhanden sind. Da an der Schnittstelle keine Operationen zu Verwaltung der ContextData vorhanden sind.

5.2.3 Project Store

Projekte und die dazugehörigen Report werden in ProjectStore persistent abgespeichert und verwaltet. Wie auch im CacheStore wird hier ebenfalls eine XML-Datenbank verwendet. Es werden alle CRUD-Operationen für die Projekte und Reporte unterstützt und zusätzlich werden Clone[9]-Operation bereitgestellt.

Zu erwähnen sind die Create-Methoden, ein CreateProject benötigt zur Instanziierung eines Projektes einen Namen sowie die Kontextinformationen der Messdatei. CreateReport die Guid des Reportes, einen Namen und ein ReportTemplate.

[9] Projekt oder Report unter anderen Name kopieren

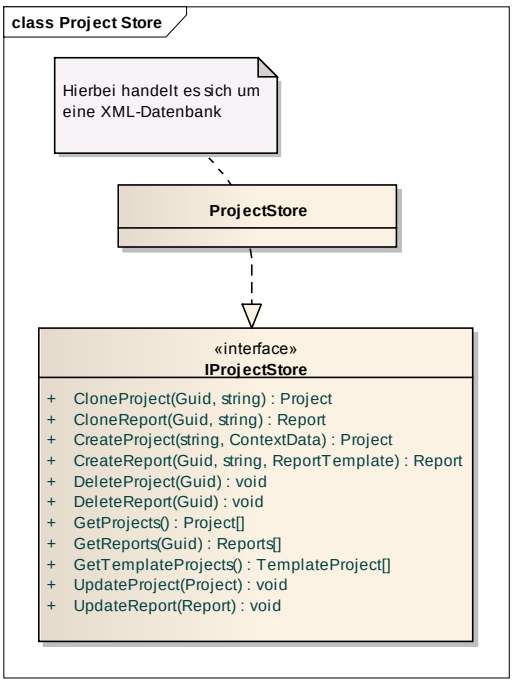

Abbildung 20 – Project Store

TemplateProject

Das Klassendiagram in Abbildung 21 zeigt die strukturierte Darstellung der Project-Klasse. Die Basisklasse für ein Projekt ist das TemplateProject, dieses wird für die automatische Generierung genutzt. Die aufgezeigten Eigenschaften sind für die Darstellung und Beschreibung eines Projekte von nützen. Wie abgebildet kann ein Template-Projekt mehrere Reporte umfassen. Dabei ist zu beachten, dass eine Komposition zwischen den beiden herrscht. Die zur Folge hat, dass beim Löschen eines Projekts alle zugehörigen Reporte ebenfalls gelöscht werden müssen. Dies muss unbedingt bei der Implementierung beachtet werden und gilt für alle Kompositionen innerhalb einer XML-Datenbank.

Project

Beim Project handelt es sich um eine Erweiterung des TemplateProjects um die Attribute ContextData und ProjectType. Das ContextData beschreibt dabei die Messung, auf welche das Projekt und die Reporte basieren.

Der ProjectTyp dient der Beschreibung des Ursprungs des Reports.

- Auto

  Das Projekt wurde durch die automatische Aggregation erstellt
- User

Vom Benutzer erstelltes Projekt

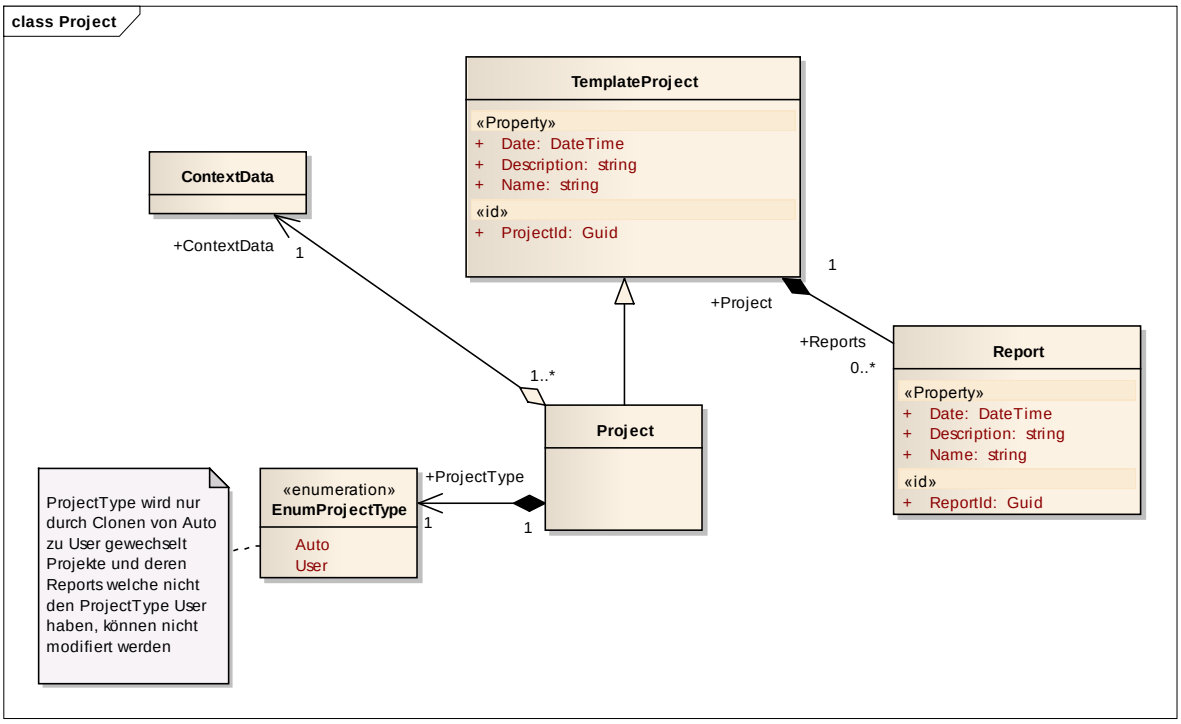

Abbildung 21 –Project

Report

Der Report beinhaltet alle wichtigen Informationen, die der Konfigurationen eines Report dienen. Zur Übersicht wurden in Abbildung 21 alle Assoziationen zum Report entfernt, diese sind unter Abbildung 4 und Abbildung 5 zu finden und müssen ebenfalls gespeichert werden.

5.3 Applikationsschicht

Die Applikationsschicht stellt die komplette Anwendungslogik zur Verfügung. Sie ist auch das Bindeglied zwischen der Benutzerschichtstelle und der Datenhaltungsschicht. Die einzelnen Komponenten innerhalb dieser Schicht sind eigenständige Server und können theoretisch auf unterschiedlichen Rechnern ausgeführt werden. In dieser Phase des Projektes laufen alle Schichten auf einem Rechner.

5.3.1 Cache Service

Eine wichtige Anforderung an die Anwendung ist ein schneller Zugriff auf vorhandene Analysen, um eine möglichst fließende Arbeit dem Benutzer zu ermöglichen. Wie unter 4.4.2.2 beschrieben, kann es durch den Einsatz einer Caching-Lösung ermöglicht werden.

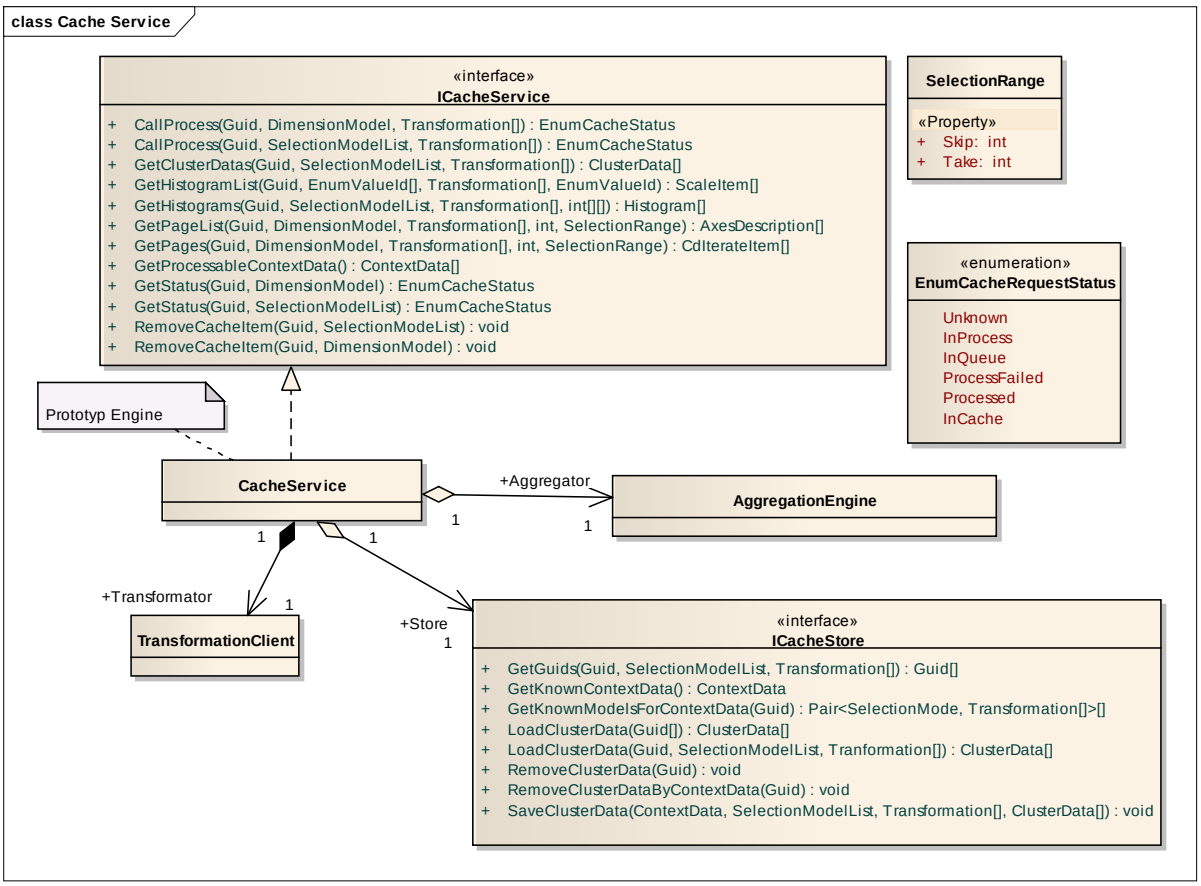

Abbildung 22 – Cache Service

Der Cache Service besteht aus einer AggregationsEngine, den CacheStore sowie einem TransformationsClient. Hierbei handelt es sich um einen vereinfachten Cache Service. Seine Aufgabe beschränkt sich auf die Überwachung der einzelnen Anfragen und der daraus entstanden Prozesse. Somit beschränkt sich das Caching nur auf die Weiterleitung, der aus dem CacheStore deserialisierten Cluster-Daten.

Die Hauptaufgaben des Cache-Services sind:

- Bereitstellung von Kontext-Informationen

  Kontext-Informationen werden direkt von der Aggregation-Engine abgefragt und an den Client zurückgesendet.
- Ausführung von Aggregationen

  Cache-Anfragen werden an die Aggregation-Engine weitergeleitet und der Status (Abbildung 23) verfolgt und entsprechend behandelt.
- Bereitstellung von Status-Informationen

  Der Status einer Anfrage wird an den Client zurückgeliefert.

Cache Status

Im Folgenden wird der Status von Cache Anfragen beschrieben:

- Unknown

  Der Status wurde initialisiert.
- InQueue

  Die Anfrage befindet sich in einer WaitingQueue.
- InProcess

  Die Anfrage wird durch den Aggregation-Server ausgeführt und befindet sich im CurrentProcessing.
- Processed

  Die Anfrage wurde ausgeführt und die Ergebnisse in den CacheStore kopiert.
- InCache

  Die Anfrage befindet sich im Cache.

  In der prototypischen Implementierung wird dem Client der InCache-Zustand auch signalisiert wenn sich dieser in Processed befindet.
- ProcessedFailed

  Die Aggregation konnte nicht vollendet werden, die Anfrage befindet sich in einer ErrorListe

Durch Entfernen[10] eines Cacheeintrags wird der Status finalisiert.

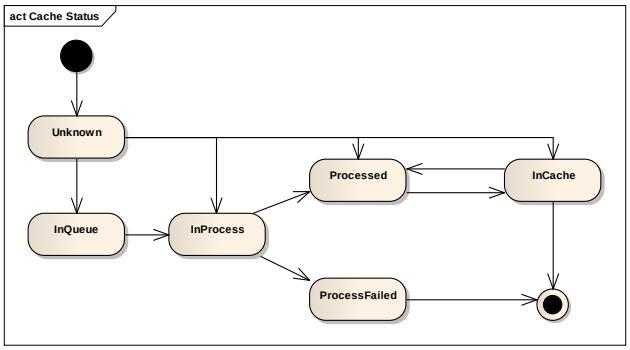

Abbildung 23 – Cache Status

CallProcess

Am Beispiel des Aufrufs der Methode CallProcess wird hier das Zusammenspiel der Komponenten des Cache-Services (Abbildung 24) aufgezeigt:

[10] RemoveCacheItem

1. CallProcess

   Der Client stellt eine Aufforderung zur Aggregation, hier muss getestet werden, ob die der Aufforderung entsprechenden Aggregationsdaten bereits im Store liegen. Oder ob ein Prozess bereits bearbeitet ist bzw. ob er in der WaitingQueue liegt. Trifft alles nicht zu, wird die Anforderung in die Queue gelegt.

2. Process

   Um eine Process am Server zu bewirken, muss dieser auf den Status Free getestet werden. Trifft dies zu, wird aus der WaitingQueue ein Antrag entnommen und in CurrentProcessing hinzugefügt. In Folge wird die Aufforderung an den Aggregator gesendet. Diese ändert seinen Status auf Processing und startet die Aggregation.

3. Aggregation

   Nach erfolgreicher Aggregation ändert der Aggregator seinen Status auf Done.

4. GetProcessedClusterData

   Der CacheService prüft am Aggregator ob sich dieser im Status Done befindet. Falls es der Fall ist, holt er die Daten vom Aggregator und dabei ändert sich der Status auf Free.

5. Speichern

   CurrentProcessing wird zurückgesetzt und die Cubes in den Store gespeichert.

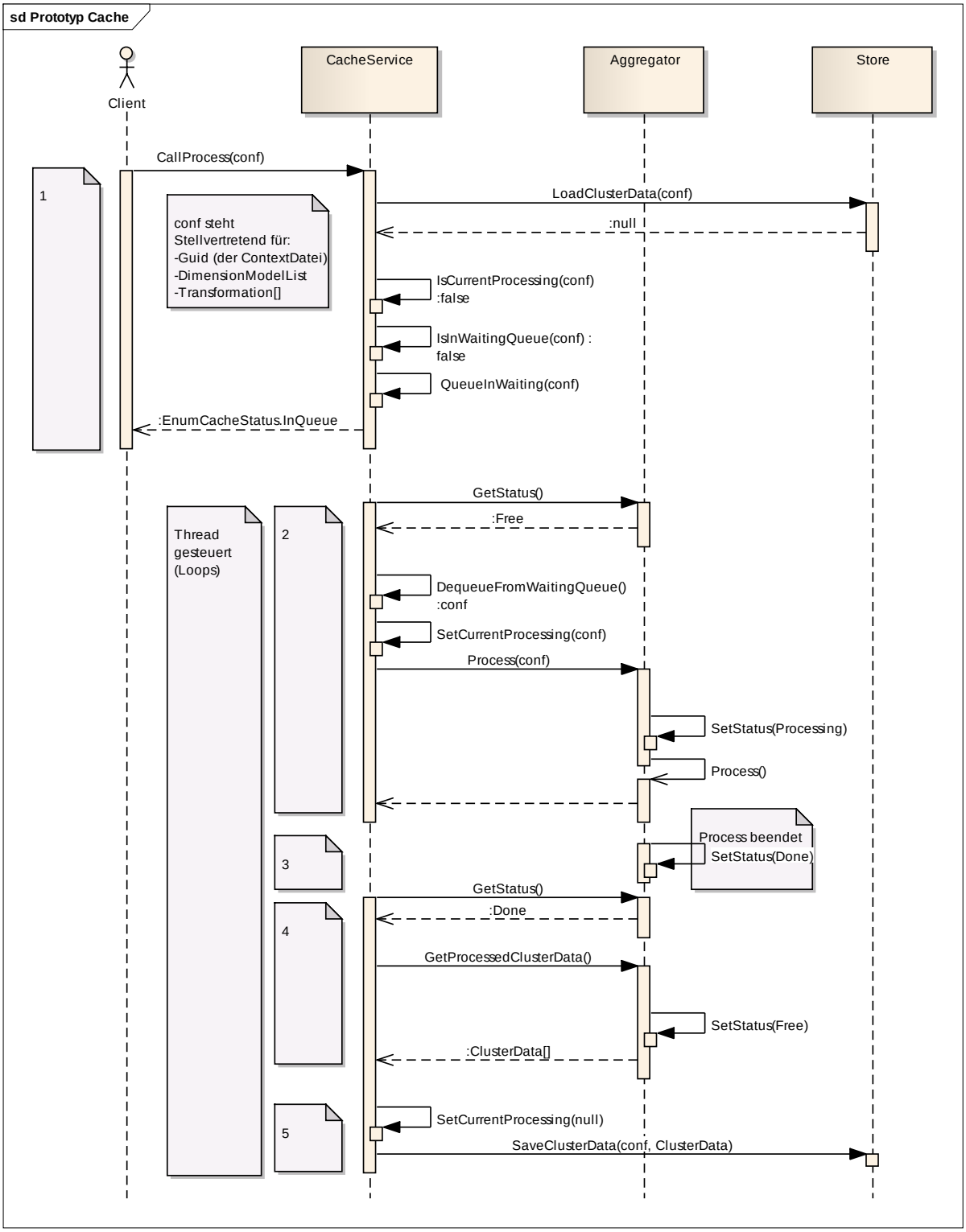

Abbildung 24 – CallProcess

5.3.2 Aggregation Engine

Die Hauptaufgabe der Aggregation-Engine ist es, eine Aggregation anhand einer Konfiguration auf einer Messdatei auszuführen.

Die Aggregation-Engine muss um die Folgenden Funktionen erweitert werden:

- GetContextData

  Liefert vorhandene Kontext-Informationen.
- GetProcessedClusterData

  Liefert die aggregierten Daten.
- GetStatus

  Gibt den derzeitigen Status der Engine an.
- Process

  Er führt die Aggregation asynchron durch, dabei wird an die SelectionModels eine Datenquelle angebunden. Hierzu wird mit der Guid, der Context-Informationen, die Messdatei im Aggregation-Store ausfindig gemacht.

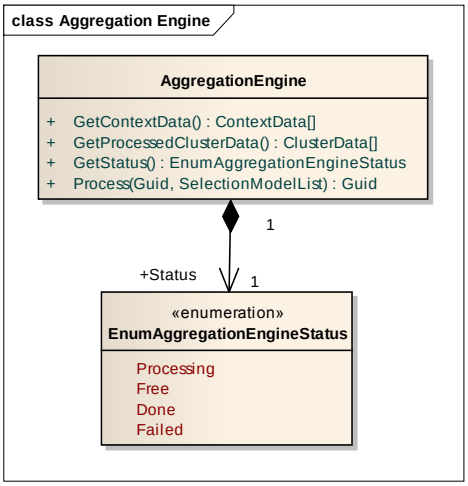

Abbildung 25 – Aggregation Engine

Die Urls zu den Messdateien (Guid) können über den Store erfragt werden.

5.3.3 Project Store Service

Der Zugriff auf die Projekte und Reporte erfolgt über den Project Store Service. Hierbei handelt es sich um einen Cache-Proxy [Fre06, 463-470].

Der Einsatz eines Proxys ist an dieser Stelle wichtig, da diese Komponente sehr viel Zugriff von der Präsentrationsschicht erhält.

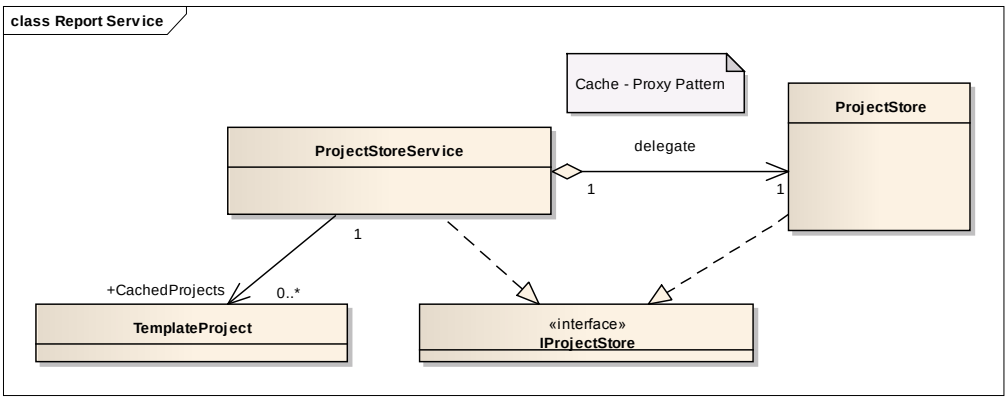

Abbildung 26 – Report Service

5.4 Präsentationsschicht

Mit der Präsentationsschicht wird eine Kommunikation zu den beiden Services hergestellt. Der Aufbau der UI entspricht dem des Project-Reports-Konzeptes. Dabei ist das User Interface in drei Bereiche eingeteilt.

Bereich 1

Dieser Bereich dient der Project Definition und Auswahl. Der Benutzer kann hier Reports erstellen, löschen oder bearbeiten. Des Weiteren kann er sich im Bereich Informationen zu den Kontextdaten holen. Bei Anklicken eines Projects ändert sich entsprechend der Inhalt im zweiten Bereich.

Bereich 2

Die Verwaltung von Reports geschieht in diesem Bereich. Der Benutzer ist hier zusätzlich in der Lage die Anfrage, welche durch den Report resultiert, an den Cache-Store zu senden. Der Status der Anfragen wird links mit Zuhilfenahme von farbigen Kreisen dargestellt.

Bereich 3

In diesem Bereich findet die eigentliche Analyse von Messdaten statt. Hier ist der Benutzer in der Lage seine Datenwürfel mittels Drag&Drop zu erstellen.

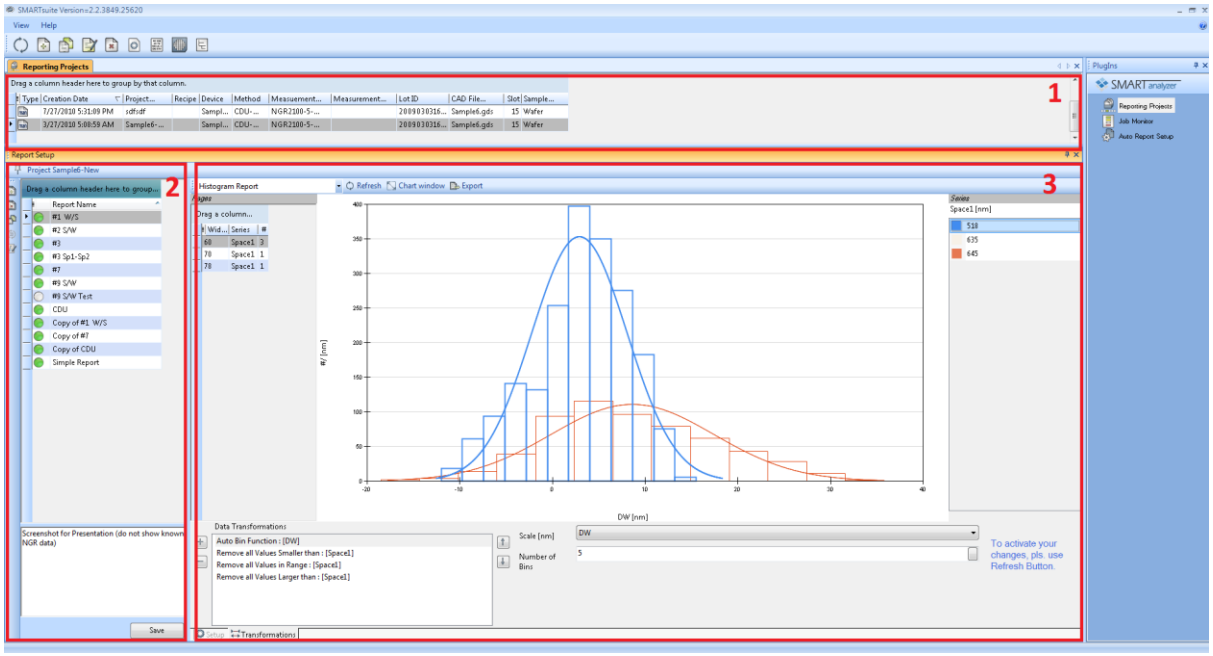

Abbildung 27 - AppFrame Übersicht

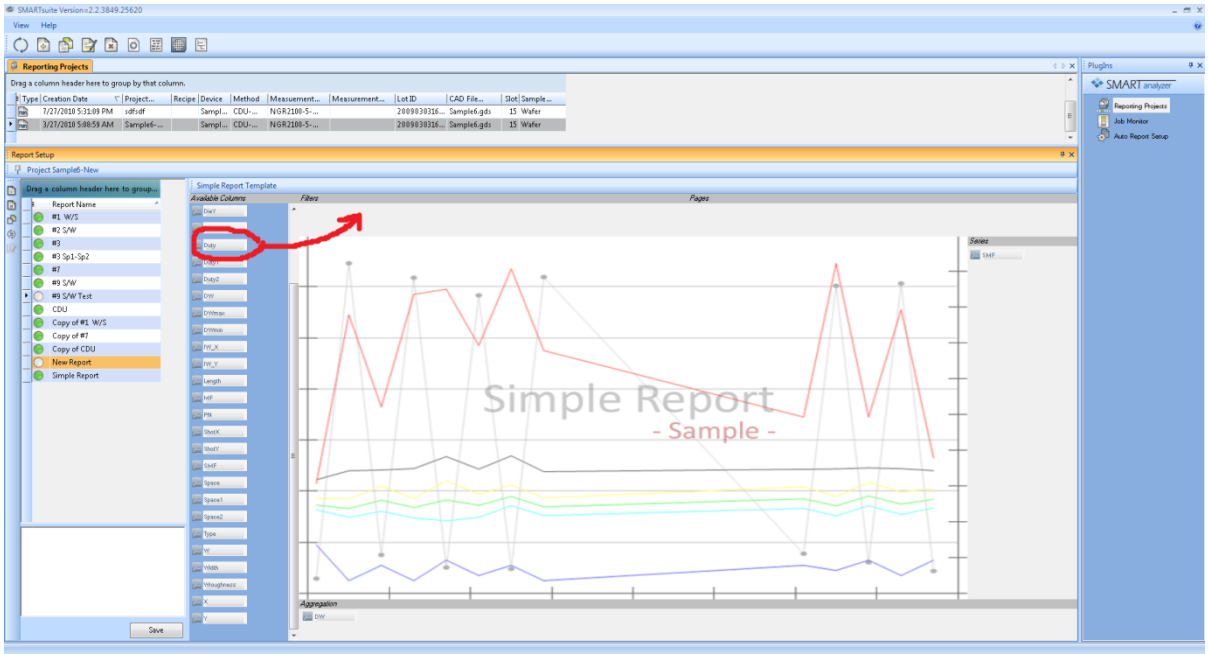

Abbildung 28 – Report Definition

6 Zusammenfassung

Die in Kapitel 2 betrachteten theoretischen Grundlagen im Bereich OLAP und Halbleiterfertigung konnten einen guten Einstieg in das Thema der Datenanalyse liefern. Auch für die später durchgeführte Analyse waren diese ein gutes Hilfsmittel. Dabei war es ein wichtiger Bestandteil der Arbeit, da sie einen sehr starken Einfluss auf den Entwurf hatten.

Das Ergebnis der Diplomarbeit ist ein funktionierender Prototyp einer Anwendung zur Massen-Messdaten Analyse. Dabei wurde der in Kapitel 5 beschriebene Entwurf umgesetzt. Es konnte jedoch nicht die komplette Funktionalität der einzelnen Komponenten realisiert werden. Dies kann jedoch Aufgrund der getrennten Komponenten unabhängig voneinander zu einem späteren Zeitpunkt implementiert werden.

Tests wurden bei der kompletten Arbeit aus zeitlichen Gründen nicht berücksichtigt, sie müssen jedoch in der nächsten Zeit nachgeholt werden. Insbesondere, Komponenten wie der Cache-Store, sind aufgrund der komplexen Implementierung recht anfällig für Fehler.

Ein Kriterium, welches in der vorliegenden Arbeit nicht direkt berücksichtig wurde, sind die Wartungs- und Hardwarekosten. Die Entwicklung und auch die Tests wurden auf einem handelsüblichen Notebook durchgeführt, somit ist dieses Kriterium ebenfalls erfüllt. Um die Wartungskosten niedriger zu halten, müsste diese Desktop-Anwendung in eine Webbasierte Anwendung umgesetzt werden.

Anhang

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading;
using System.IO;
using System.Xml.Serialization;
using System.Runtime.Serialization.Formatters.Soap;
using System.Runtime.Serialization;

namespace SerializerTest
{
    class Program
    {
        static int fact = 10000;

        static void Main(string[] args)
        {
            var soap = new SoapFormatterWrapper();
            var bin = new BinaryFormatterWrapper();
            var xml = new XmlSerializerWrapper(typeof(ServerResponse[]));
            var net = new NetDataContractSerializerWrapper();
            var dataO = new DataContractSerializerWrapper(typeof(ServerResponse[])
                                                          ,false);
            var dataR = new DataContractSerializerWrapper(typeof(ServerResponse[])
                                                          ,true);

            // Umleitung von StdOut auf Datei
            var writer = new StreamWriter(@"c:\temp\serTest_ServerRes.csv")
                        {
                            AutoFlush = true
                        };
            Console.SetOut(writer);

            var s = ";binSer;binDeser;binSize;xmlSer;"+
                "xmlDeser;xmlSize;soapSer;soapDeser;soapSize;"+
                "netSer;netDeser;netSize;dataSer;dataDeser;dataSize;"+
                "dataRSer;dataRDeser;dataRSize";
            Console.WriteLine(s);
            for (int round = 0; round < 100; round++)
            {
                //var containers = new int[fact * round];
                var containers = new ServerResponse[fact * round];

                for (int i = 0; i < fact * round; i++)
                {
                    containers[i] = new ServerResponse
                                    {
                                        Reponse = EnumResponse.Busy,
                                        Value = new double[i]
                                    };
                }
```

```
            Console.Write(fact * round);
            Export(bin, containers);
            Export(xml, containers);
            Export(soap, containers);
            Export(net, containers);
            Export(dataO, containers);
            Export(dataR, containers);

            Console.WriteLine();
        }

        writer.Flush();
        writer.Close();

        var std = new StreamWriter(Console.OpenStandardOutput());
        std.AutoFlush = true;
        Console.SetOut(std);
        Console.WriteLine("..done");
        Console.ReadKey();
    }

    public static void Export(ISerializeWrapper formatter, object content)
    {
        const string fileName = "test.bin";
        var target = new FileStream(fileName, FileMode.Create);

        var start = DateTime.Now;
        formatter.Serialize(target, content);
        var serTime = (DateTime.Now - start).TotalSeconds;

        target.Position = 0;
        start = DateTime.Now;
        formatter.Deserialize(target);
        var deserTime = (DateTime.Now - start).TotalSeconds;

        target.Dispose();

        var fi = new FileInfo(fileName);

        Console.Write(";" + serTime + ";" + deserTime  + ";" +
                                            fi.Length / 1024d);
    }

    public interface ISerializeWrapper
    {
        void Serialize(Stream stream, object content);
        object Deserialize(Stream streamfileStream);
    }

    public class SoapFormatterWrapper : ISerializeWrapper
    {
        private readonly SoapFormatter _ser;

        public SoapFormatterWrapper()
        {
            _ser = new SoapFormatter();
        }
```

```
        public void Serialize(Stream stream, object content)
        {
            _ser.Serialize(stream, content);
        }

        public object Deserialize(Stream stream)
        {
            return _ser.Deserialize(stream);
        }
    }

    public class BinaryFormatterWrapper : ISerializeWrapper
    {
        private readonly BinaryFormatter _ser;

        public BinaryFormatterWrapper()
        {
            _ser = new BinaryFormatter();
        }

        public void Serialize(Stream stream, object content)
        {
            _ser.Serialize(stream, content);
        }

        public object Deserialize(Stream stream)
        {
            return _ser.Deserialize(stream);
        }
    }

    public class XmlSerializerWrapper : ISerializeWrapper
    {
        private readonly XmlSerializer _ser;

        public XmlSerializerWrapper(Type t)
        {
            _ser = new XmlSerializer(t);
        }

        public void Serialize(Stream stream, object content)
        {
            _ser.Serialize(stream, content);

        }

        public object Deserialize(Stream stream)
        {
            return _ser.Deserialize(stream);
        }
    }

    public class NetDataContractSerializerWrapper : ISerializeWrapper
    {
        private readonly NetDataContractSerializer _ser;

        public NetDataContractSerializerWrapper()
        {
            _ser = new NetDataContractSerializer();
```

```
            }

            public void Serialize(Stream stream, object content)
            {
                _ser.Serialize(stream, content);
            }

            public object Deserialize(Stream stream)
            {
                return _ser.Deserialize(stream);
            }
        }

        public class DataContractSerializerWrapper : ISerializeWrapper
        {
            private readonly DataContractSerializer _ser;

            public DataContractSerializerWrapper(Type t, bool objRefs)
            {
                _ser = new DataContractSerializer(t, new[]
                        {
                            typeof(int),
                            typeof(string),
                            typeof(ServerResponse),
                        }, int.MaxValue, false, objRefs, null);
            }

            public void Serialize(Stream stream, object content)
            {
                _ser.WriteObject(stream, content);
            }

            public object Deserialize(Stream stream)
            {
                return _ser.ReadObject(stream);
            }

        }


    }

    [Serializable]
    public class ServerResponse
    {
        public EnumResponse Reponse { get; set; }
        public double[] Value { get; set; }
    }

    [Serializable]
    public enum EnumResponse
    {
        Busy,
        Waiting,
        LongProcess
    }

}
```

Literaturverzeichnis

[Bal99] Balzert, Heide. *Lerhbuch der Objektmodellierung; Analyse und Entwurf.* 3827402859 Bde. Spektrum Akademischer Verlag GmbH, 1999.

[Ben04] Bengel, Günter. *Grundkurs Verteilte Systeme.* Vieweg Verlag, 2004.

[Bod06] Bodendorf, Freimut Bodendorf. *Daten- und Wirtschaftsmanagement.* Springer-Lehrbuch, 2006.

[Fre06] Freemann, Eric. *Entwurfsmuster von Kopf bis Fuss.* 2006.

[Doo01] Kevin, Dooley. *Designig Large Scale Lans.* O'Reilly Media, Inc, 2001.

[Kle10] Klein, Scott. *Pro Entity Framework 4.0.* apress, 2010.

[Kur99] Kurz, Andeas. *Data Warehousing, Enabling Technology.* Bonn: MITP-Verlag, 1999.

[Mas05] Masak, Dieter. *Moderne Enterprise Architekturen.* Springer-Verlag, 2005.

[Nag06] Nagabhushana, S. *Data Warehousing OLAP and Data Mining.* New Delhi: New Age International Pvt., 2006.

[Pis84] Pissanetzky, Sergio. *Sparse Matrix Technology.* ACADEMIC PRESS, 1984.

[Ras06] Rasch, Björn, Malte Friese, Wilhelm Hofmann, und Ewald Naumann. *Quantitative Methoden Band 1.* Springer Medizin Verlag, 2006.

[Tab05] Tabeling, Peter. *Softwaresysteme und ihre Modellierung: Grundlagen, Methoden und Techniken.* Springer, 2005.

[Tho02] Thomsen, Erik. *OLAP Solutions - Building Multidimensional Information Systems.* John Wiley & Sons, Inc., 2002.

[WWW01] Serialization, MSDN - Microsoft. 2010. http://msdn.microsoft.com/en-us/library/ms233843(v=VS.100).aspx (Zugriff am 28. November 2010).

[WWW02] FASMI, *http://www.bi-verdict.com/fileadmin/FreeAnalyses/fasmi.htm* (Zugriff am 28. November 2010).

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

……

Dresden, 30. November 2010